An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols

THESIS

Karl R. Schrader, Major, USAF

AFIT/GCE/ENG/09-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCE/ENG/09-10

An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Karl R. Schrader, BSEE, MAS

Major, USAF

March 2009

AFIT/GCE/ENG/09-10

An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols

Karl R. Schrader, BSEE, MAS
Major, USAF

Approved:

Barry Mullins	12 Feb 09
Dr. Barry E. Mullins (Chairman)	date
[signature]	12 FEB 09
Dr. Gilbert L. Peterson (Member)	date
Robert J Mills	12 Feb 09
Dr. Robert F. Mills (Member)	date

Abstract

This thesis addresses the problem of identifying and tracking digital information that is shared using peer-to-peer file transfer and Voice over IP (VoIP) protocols. The goal of the research is to develop a system for detecting and tracking the illicit dissemination of sensitive government information using file sharing applications within a target network, and tracking terrorist cells or criminal organizations that are covertly communicating using VoIP applications.

A digital forensic tool is developed using an FPGA-based embedded software application. The tool is designed to process file transfers using the BitTorrent peer-to-peer protocol and VoIP phone calls made using the Session Initiation Protocol (SIP). The tool searches a network for selected peer-to-peer control messages using payload analysis and compares the unique identifier of the file being shared or phone number being used against a list of known contraband files or phone numbers. If the identifier is found on the list, the control packet is added to a log file for later forensic analysis.

Results show that the FPGA tool processes peer-to-peer packets of interest 92% faster than a software-only configuration and is 99.0% accurate at capturing and processing BitTorrent Handshake messages under a network traffic load of at least 89.6 Mbps. When SIP is added to the system, the probability of intercept for BitTorrent Handshake messages remains at 99.0% and the probability of intercept for SIP control packets is 97.6% under a network traffic load of at least 89.6 Mbps, demonstrating that the tool can be expanded to process additional peer-to-peer protocols with minimal impact on overall performance.

Acknowledgements

I would like to thank my thesis adviser, Dr. Barry Mullins, for providing the right balance of guidance to keep me on track and freedom to explore my ideas and bring them to fruition. In addition, I would also like to thank Dr. Gilbert Peterson and Dr. Robert Mills for their support and assistance.

I greatly appreciate the help of Major David Olander, whose insight and assistance made the process of learning how to program and implement the experimental system a less painful experience. I would also like to thank Captain Benjamin Ramsey for reminding me that Voice over IP is also a peer-to-peer protocol.

Finally, I would be remiss without thanking my wife. Her continuous support and understanding were priceless throughout this process.

Karl R. Schrader

Table of Contents

Page

List of Figures

List of Tables

List of Abbreviations

An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols

I. Introduction

1.1 Motivation

Peer-to-peer (P2P) networking has changed the way users search for, send, and receive digital information over the Internet. Instead of relying on interactions with centralized servers to upload and download digital content, users now share music, movies, documents, and conversations directly with other users. While peer-to-peer networking provides new and powerful applications for the legitimate distribution of digital information, it is also being used for many illicit purposes as well.

One high-profile illicit use of peer-to-peer networking technology is for the dissemination of child pornography. The Federal Bureau of Investigation's (FBI) Regional Computer Forensics Laboratory states in its 2007 annual report that "cybercrime, which includes crimes against children and child pornography, is the offense for which law enforcement requested assistance most often" [oJ08]. In addition, a 2005 Government Account Office report stated that "[Peer-to-peer] technology is increasingly popular for disseminating child pornography" [Off05].

Another area in which peer-to-peer networking is being used for illegal activities is covert communication among terrorist cells through Voice over IP (VoIP) protocols. In Afghanistan, the Taliban has begun using VoIP-based Internet phones to coordinate attacks on Coalition forces while avoiding detection. In fact, according to one British Government official, VoIP calls are "seriously undermining" MI6's ability to intercept and track Taliban communications [Owe08]. In addition, during the December 2008 terrorist attacks in Mumbai, India, the attackers' Pakistan-based handlers sent instructions, intelligence, and encouragement using VoIP-based Inter-

net phones [Kah08]. During the 3-day series of attacks, the terrorists were able to communicate without their calls being traced or intercepted by authorities.

To help combat these illicit uses for peer-to-peer networking, the goal of this research is to develop a system to identify and track any type of digital information that is transmitted on a network using peer-to-peer protocols. Several methods have already been developed to accomplish this task and are in use today, but they depend on the use of honeypots to lure targets into downloading contraband material [BSCF07], physical access to the suspected file sharer's computer [AJ07], active searching of the Internet for contraband files to download [CCM+07], or active interception and modification of contraband file sharing requests [DS08a]. All of these methods are active attempts to discover illicit file sharing, with the drawback that they can all be detected and possibly circumvented by file sharers that are aware of their presence. In contrast, the system developed for this research consists of a suite of tools that passively detects and tracks illicit file sharing on a target network without affecting the flow of traffic on the network, making it impossible for users of the network to determine the presence of the system.

This system is useful for law enforcement, intelligence agencies, and computer network system administrators across the US Government. Law enforcement agencies can use these tools to identify child pornography being transmitted across a network and track both the sender and receiver to their sources. Intelligence agencies can use the system to track known terrorists using VoIP technology for their communications. System administrators can also use the tools to detect the unauthorized transmission of sensitive files from their networks and determine the sender and recipient, thereby identifying potential insider threats and inadvertent disclosures of sensitive information.

1.2 Overview and Goals

This thesis focuses on designing and constructing a digital forensic tool, called the TRacking and Analysis for Peer-to-Peer (TRAPP) system, that allows an inves-

tigator or system administrator to monitor network traffic in real-time for any digital information that meets the user's definition of contraband being shared using peer-to-peer protocols. The TRAPP system, shown in Figure 1.1, is designed to be set up on the gateway between a Government-owned network and the Internet. As packets pass through the gateway, copies are sent to the system for analysis. For each packet received, TRAPP inspects the packet to determine if it is a control packet for a peer-to-peer protocol of interest. If the packet is not a peer-to-peer control packet, it is discarded. If the packet is a control packet, the system extracts from the packet's payload the unique identifier for the data being shared, and attempts to match the identifier against a list of files of interest in the system's memory. If a match is not made, the packet is discarded. If a match is made, the control packet is recorded in a log file for future analysis.

There are three primary goals for this research. The first goal is to construct a hardware-based system using a Field Programmable Gate Array (FPGA) that analyzes all network traffic sent to it, detects packets belonging to a specific peer-to-peer protocol, compares the digital information being shared against a list of interest, and in the case of a match, records selected control packets from the peer-to-peer session in a log file. The second goal is to optimize the system to increase the probability of detecting and recording all control packets, even when network traffic is being sent to the system at nearly the full capacity of the system's Ethernet controller. The last goal is to demonstrate the system's expandability by modifying it to accept an additional peer-to-peer protocol with no impact on overall performance.

1.3 Thesis Layout

This chapter introduces the research topic, provides a motivation for the research efforts, and outlines the goals of the research. Chapter 2 presents background information on the types of peer-to-peer networks, related work in the area of detecting illicit file sharing, methods of classifying network traffic, and ways to obfuscate peer-to-peer traffic. The BitTorrent peer-to-peer protocol is discussed in Chapter 3,

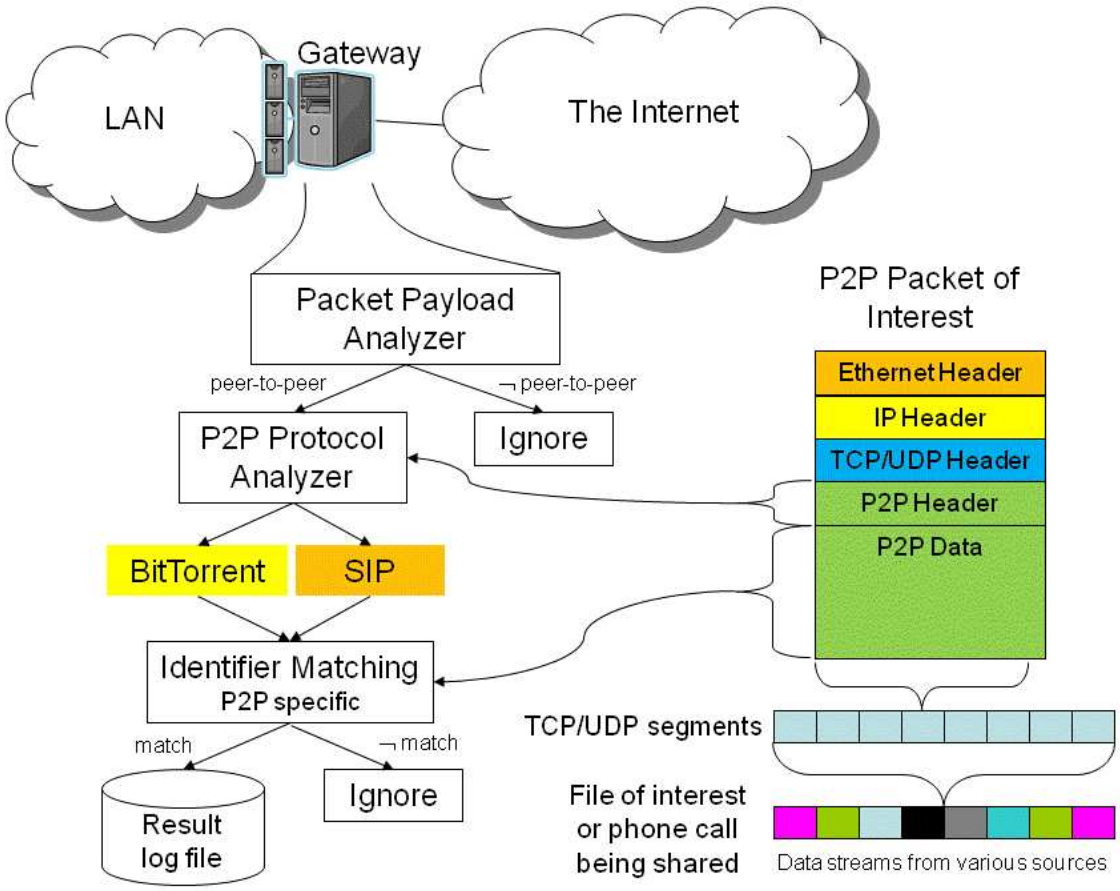

Figure 1.1: The Proposed TRAPP System

and the Session Initiation Protocol is discussed in Chapter 4. Chapter 5 outlines the methodology used to design, set up, and conduct the experiments to test the effectiveness of the TRAPP system. Chapter 6 provides a discussion and analysis of the experimental results. The conclusions drawn from the experimental results, the significance of the completed TRAPP system, and areas for future research are given in Chapter 7. Appendix A describes how to construct the hardware portion of the TRAPP system, and Appendix B contains the raw data collected during the experiments.

II. Literature Review and Related Research

This chapter presents an overview of background information and related research on peer-to-peer protocols, detection of illicit file sharing, Internet traffic classification, and peer-to-peer traffic obfuscation techniques. Section 2.1 provides historical background on several popular peer-to-peer network protocols. Related research in the area of identifying and tracking illegal file downloaders is presented in Section 2.2. Sections 2.3 and 2.4 detail traditional methods and recent work on nontraditional methods of classifying Internet traffic. Section 2.5 outlines the current methods of obfuscating and encrypting peer-to-peer traffic, and the chapter is summarized in Section 2.6.

2.1 Background On the Use of Peer-to-Peer Traffic

Prior to 1999, the Internet was primarily used for World Wide Web surfing, email, and FTP file transfers [The06]. Then, a new method of transferring files quickly and easily, called peer-to-peer networking, was introduced. Since then, peer-to-peer networking has steadily grown in popularity and is now the dominant component of worldwide Internet traffic [P2P07]. Described below is a short history of the rise of peer-to-peer networking followed by some statistics on the use of peer-to-peer networking on the Internet.

2.1.1 The Rise and Fall of Napster. The first widely used peer-to-peer networking service was Napster, which was created by Shawn Fanning in 1999 for the purpose of sharing .mp3 music files [Tys08]. In the Napster architecture, users download the Napster client, install it, and then connect to centralized servers, run by the Napster Corporation. These servers contain a constantly-updated list of all files currently available on client computers. When a user wants to download a file, he queries the server for IP addresses of other peers who possess the file, and then connects directly with a peer for the file transfer [Tys08].

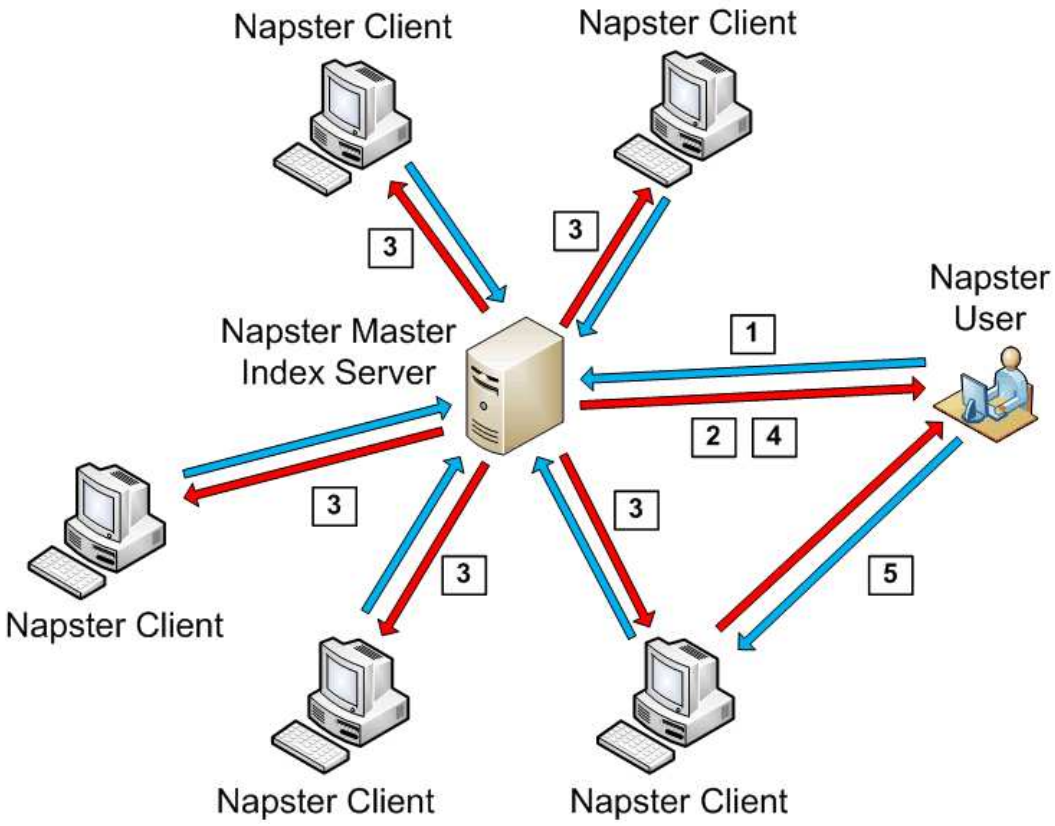

Figure 2.1: How to Download a File Using the Napster Network

Figure 2.1 shows how the Napster system is used to download a file. As shown in the figure, when downloading a file, the following occurs:

1. The client connects to the central cluster of Napster servers to search for a file to download.
2. While the client is connected to the server, the server in return searches the client for sharable files, and the client's file directory is added to the server's list.
3. The server searches its indexes for files matching the client's request. These indexes are compiled by querying clients currently connected to the server and retrieving their file directories.
4. The Napster server sends the client a list of peers who have the file.

5. The client connects to the peer that has the file and downloads it. Note that this exchange happens without the server in the loop [Fel04].

While the Napster servers did not contain any illegal or copyrighted files, the corporation did suffer from a serious legal vulnerability. Even though Napster did not directly provide illegal files to clients, it did keep centralized records, under its control, of every file being transferred. The Recording Industry Association of America and the major music labels sued Napster, claiming that it allowed illegal activities to occur with its full knowledge and consent. These lawsuits resulted in the closure of Napster in 2001 [Fel04].

2.1.2 The Rise of Decentralized Peer-to-Peer Systems. In order to avoid the legal pitfall that befell the Napster system, almost all newer peer-to-peer protocols eliminate the centralized server model in favor of a decentralized model using super-peers or supernodes. For example, in the Fast Track/KaZaA protocol, certain clients are designated as supernodes or superpeers, and are responsible for maintaining and distributing searchable lists of files. Every client possesses the software to become a supernode, and any client can be promoted to a superpeer at any time. This allows the corporations distributing the software to avoid lawsuits, since they now have no knowledge of what is being shared; that information resides on the supernodes, which are not under the control of the corporation [Fel04].

In another implementation of decentralized peer-to-peer networking, the Gnutella protocol uses query flooding to search for and download files. Figure 2.2 shows how the Gnutella system is used to download a file [The08]. As shown in the figure, when downloading a file, the following occurs:

1. When a client wishes to search for a file, he sends the query to a small number of other clients he is connected to (usually less than five), asking for a file.
2. Each of those clients then sends the query to the peers that it is connected to, and so on, up to a maximum number of hops away from the original requester.

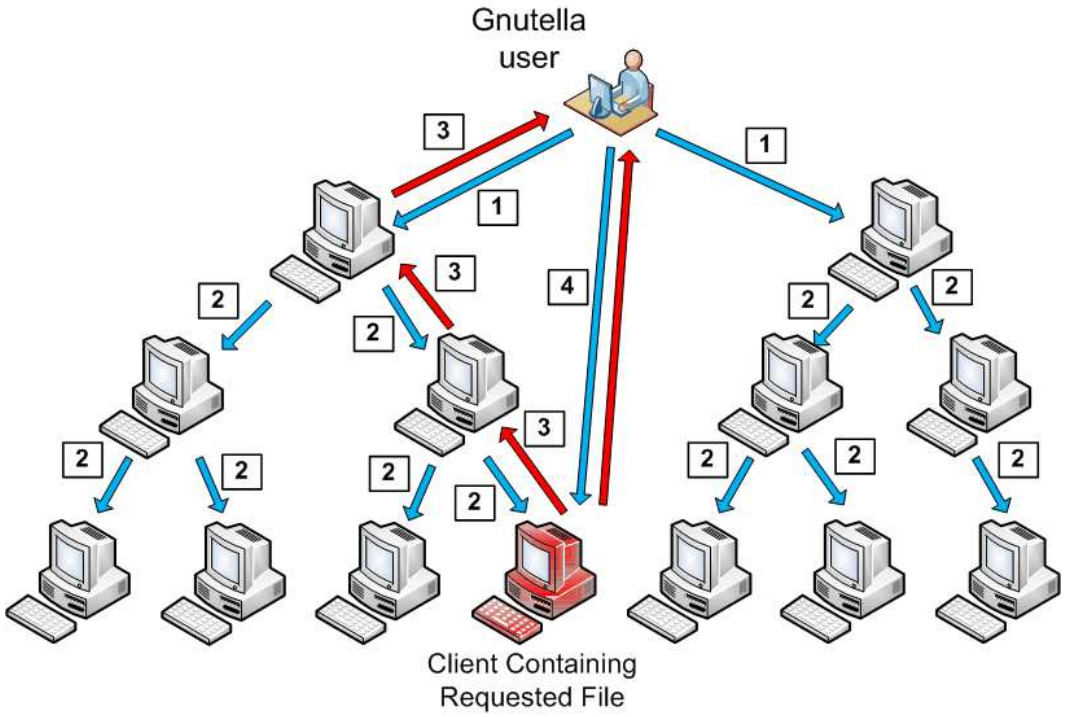

Figure 2.2: How to Download a File Using the Gnutella Network

3. If another client has the file, it sends a positive reply back to the requester using the reverse of the route used by the requester's query.

4. A direct connection is set up between the requester and the client containing the requested file, and the file is transferred using the HTTP protocol.

For this research, the focus is on the BitTorrent protocol, another decentralized peer-to-peer protocol, which is discussed in detail in Chapter 3.

2.1.3 Statistics on P2P Traffic Usage on the Internet. The use of the Internet for peer-to-peer file sharing has steadily increased since the introduction of the Napster protocol in 1999. In 2000, a University of Wisconsin study found that Napster traffic had supplanted HTTP traffic as the dominant protocol in use on the university's network [Plo00]. In 2002, the University of Washington determined that peer-to-peer traffic accounted for 43% of all university traffic, with only 14% of traffic devoted to HTTP [SGD+02]. In a 2007 study, Ipoque, a German-based company

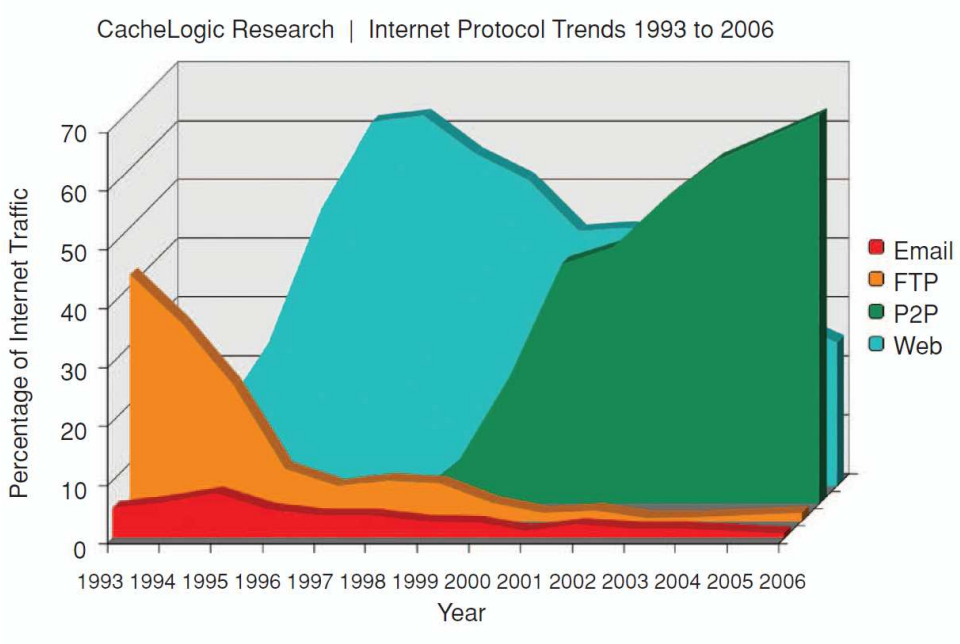

Figure 2.3: Peer-to-peer Networking as a Percentage of Total Internet Traffic [The06]

that specializes in network management for Internet service providers, determined that between 49 and 83 percent of all Internet traffic is peer-to-peer-related [P2P07]. In January 2006, Cachelogic performed a study that showed approximately 61% of all Internet traffic is peer-to-peer related, compared to only 32% for HTTP [The06]. Figure 2.3 graphically shows this massive increase in the use of peer-to-peer networking. To put this into perspective, a survey conducted in 2005 found that peer-to-peer networks worldwide contained over 2.8 billion files available for download [CCM+07].

2.2 Current Methods of Identifying Downloaders of Illegal Files

Given the rapid rise of peer-to-peer file sharing, law enforcement agencies and copyright holders are struggling to keep up with illegal file sharers. Currently, there are several methods available to these entities to identify and track illegal file downloaders. These methods include the use of honeypots to lure suspects into downloading illegal files provided as bait by law enforcement, physically searching a suspect's computer for illegal files that have been downloaded in the past, identifying illegal

downloads on BitTorrent through the exhaustive search of files available for download, and a new method of active interception and modification of contraband file sharing requests.

2.2.1 Honeypots. One simple method of identifying and tracking uploaders and downloaders of contraband files is through the use of honeypots. In the context of this discussion, a honeypot is a trap designed to detect and track illegal activities. In its most basic execution, a computer is set up on the Internet with a collection of illegal files. These files are then advertised on various peer-to-peer networks, where they await downloading by illegal file sharers. When another computer attempts to download the illegal files, the downloader's IP address, port number, date, time, and the packets being downloaded are recorded by the honeypot owner. This information is then either sent to law enforcement agencies for prosecution or used as evidence in civil lawsuits brought by the owners of the files being downloaded.

In their research, Badonnel, State, Christment, and Festor design and test a management platform for tracking illegal file sharers using peer-to-peer networks [BSCF07]. Their platform contains a configurable honeypot agent that is "capable of advertising parameterizable undesirable content and of identifying cyberpredators that request those contents." To create and track these agents, they also create a separate "platform manager" that configures the honeypot agents and organizes the resulting data, including the list of illegal file downloaders that the agents have detected and tracked. A custom network protocol is used for communication between the agents and the manager.

Figure 2.4 shows how the agents and management platform interact to perform this detection and tracking. To successfully capture a cyberpredator, the system performs the following functions [BSCF07]:

1. *Set Configuration.* The platform manager configures the honeypot agent. When the manager first deploys the honeypot agents, it provides those agents with the list of keywords used to generate the bait files, along with a set of rules for how

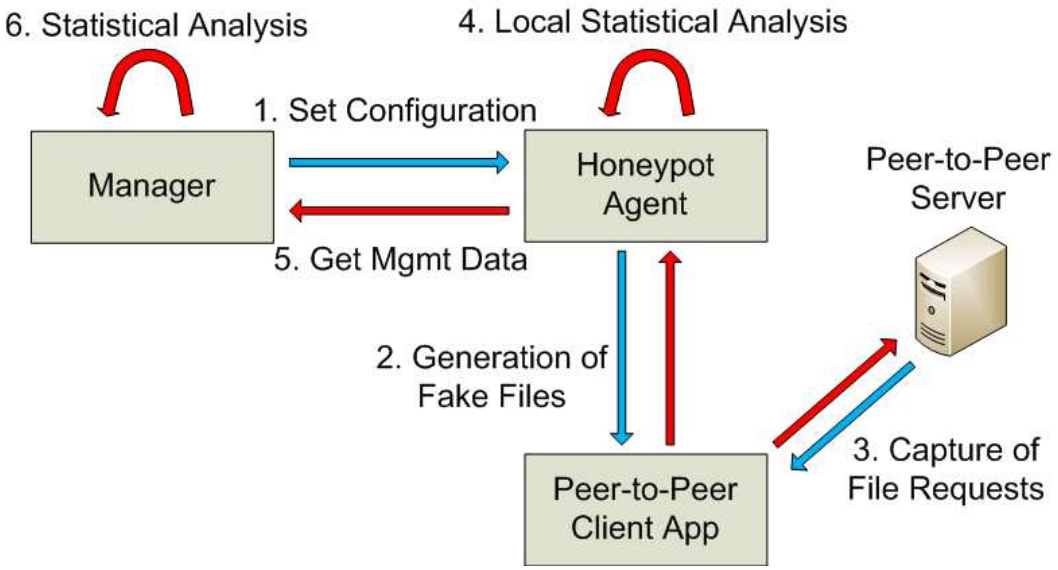

Figure 2.4: How Honeypot Agents and the Platform Manager Interact to Detect and Track Illegal File Downloaders (Adapted from [BSCF07])

the files will be generated and the content of the files. The manager also provides the initial set of rules that the agent will employ in its local statistical analysis of downloading activity.

2. *Generation of Fake Files.* The proper generation of file names by the honeypot agent is extremely important to the effectiveness of the honeypot. To generate the fake illegal files, the agent is given a list of keywords common to the types of files being offered. The agent then randomly generates file names using combinations of the keywords and attaches extensions corresponding to the most common formats of illegally download files, such as audio, picture, and video file types.

3. *Capture of File Requests.* The honeypot agent must accurately detect, identify, and record all attempts to download the illegal files that it generated in the step above. The agent is capable of analyzing all inbound traffic and identifying sessions using a peer-to-peer protocol. Once the sessions are identified, the agent records the files being downloaded as well as the identifying information of the downloader, such as IP address and port number.

4. *Local Statistical Analysis.* The honeypot agent performs a statistical analysis of the data collected, and identifies information such as the most requested file keywords and the most active downloaders. The agent uses these results to determine potential patterns of downloading by certain users and to refine the generation of file names for future download.

5. *Get Management Data.* The platform manager gets management data from the honeypot agent. The manager uses the custom platform communication protocol to retrieve the results of the local statistical analyses from the honeypot agents. These results include the list of most request files, most searched for keywords, and the most active downloaders from the agent's local perspective. These results are organized and stored in a database for further analysis.

6. *Statistical Analysis.* The platform manager performs its own statistical analysis. The manager takes the local statistical data from all the honeypot agents and performs a platform-wide statistical analysis on the data. This integrated analysis allows the manager to identify illegal file downloaders that initially escape notice by the local agents because of their downloading patterns. In addition, the analysis also provides the manager with data on what keywords are most often searched for platform-wide, which is then used to update the file generation keyword lists that are provided to individual honeypot agents.

2.2.1.1 Shortcomings of the Honeypot Method. While effective against illegal downloaders who access the honeypots, there are several shortcomings to using this method for identifying and tracking illegal downloaders.

First, the illegal downloader has to access the honeypot. While a seemingly trivial problem, the advent of IP blacklists among the downloading community has dealt a major setback to the honeypot method. An IP blacklist is a list of IP addresses that are known or suspected to be used by government authorities and corporations as sources of honeypots to trap and track downloaders. Today, programs such as Peer Guardian are specifically designed to act as a downloading firewall that blocks these

blacklisted IP addresses, preventing the user's peer-to-peer software from downloading from them [Gil08].

Second, honeypots are an active method of detection, in that file sharers must actively download from the honeypot in order to be identified by law enforcement agencies. For certain classes of highly illegal files, such as child pornography, hard to find and private invite-only websites are used to keep the general public and law enforcement from accessing and downloading them [Mac06]. It is very difficult for honeypots to become registered peers on these private sites, further decreasing their usefulness.

2.2.2 Hardware Recovery of Illegal Files. Another method of identifying potential illegal file downloaders is to search the suspect's computer for illegally downloaded files using digital forensic techniques. In their research, Adelstein and Joyce introduce a digital forensic tool called File Marshal which allows law enforcement to automatically detect and analyze peer-to-peer software usage on a hard drive [AJ07]. Figure 2.5 shows the four phases used by the File Marshal software to recover peer-to-peer usage data [AJ07].

2.2.2.1 Discovery Phase. In the discovery phase, the File Marshal software analyzes the target hard drive and searches for any peer-to-peer client software that is currently or was previously installed. By searching for telltale signs of peer-to-peer software in the system registry, file system directory structure, path statements, and data directories, the software can identify what peer-to-peer software has been used on the computer, even programs that were uninstalled prior to the search.

2.2.2.2 Acquisition Phase. In the acquisition phase, the software retrieves all configuration and log information from the peer-to-peer software applications. These configuration and log files contain useful information such as the names of downloaded files, when the files were downloaded, what servers the peer-to-peer

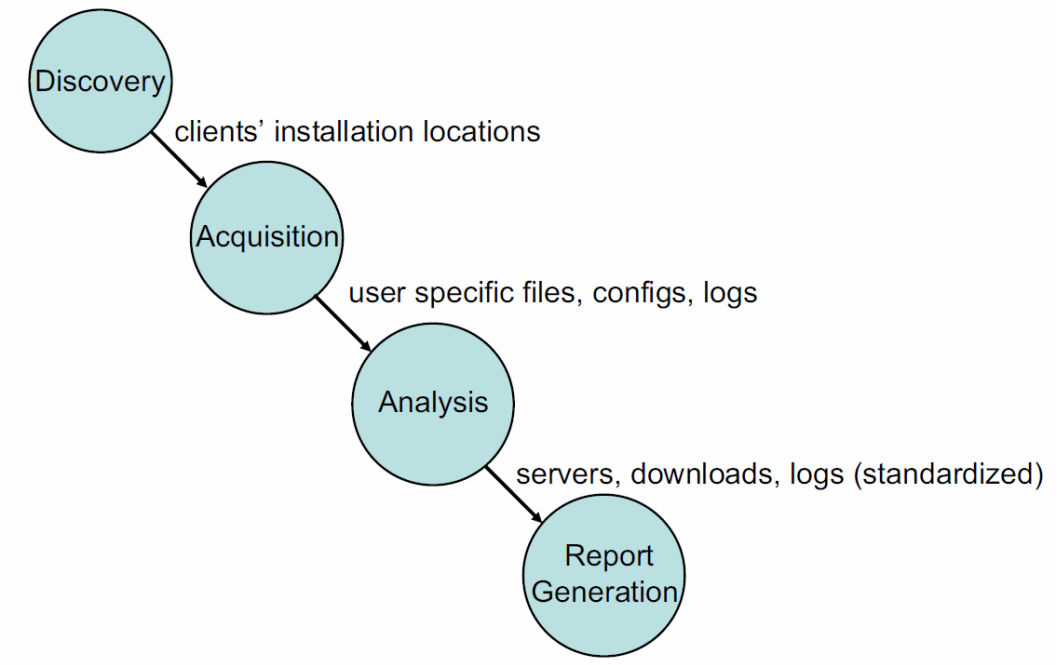

Figure 2.5: The File Marshal Forensic Software Investigation Process [AJ07]

client has connected to in the past, and lists of hashes for files that have been uploaded and downloaded.

2.2.2.3 Analysis Phase. In the analysis phase, the software repackages the information retrieved in the acquisition phase into a format that is easily read and manipulated by the forensic investigator. File Marshal allows the investigator to view downloaded files, sort lists by various parameters, and search for files by providing certain criteria, such as a specific hash value.

2.2.2.4 Report Generation Phase. The last phase, report generation, involves the software providing an audit trail of every step taken by the forensic investigator and by the File Marshal tool for authentication and verification purposes. The software also creates summary reports that can easily be imported into a larger forensic report.

2.2.2.5 Shortcomings of the Hardware Recovery Method. As with the use of honeypots, there are several drawbacks to the hardware recovery method. First, the investigator must physically possess the hard drive. In most cases, this requires some kind of legal action to force a suspect to turn over his computer to the investigator, which can be an extremely invasive procedure.

Second, in order to recover a suspect's hard drive for analysis, investigators must first determine that the hard drive is worth analyzing. In other words, investigators must already suspect the computer as containing illegal files before acting to confiscate the drive for analysis. This type of investigation is extremely time and labor intensive, limiting the ability of law enforcement to tackle widespread illegal file sharing using this method.

2.2.3 The BitTorrent Monitoring System. A specialized method for detecting and tracking illegal file downloaders is the BitTorrent Monitoring (BTM) system, designed by Chow, et. al. [CCM+07]. BTM is a system that automatically searches the Internet for BitTorrent-based downloadable files, analyzes the files to determine if they are illegal, attempts to download the suspected illegal files, and finally records tracking information on who provided the files for download. For more information on the BitTorrent peer-to-peer protocol, see Chapter 3.

Figure 2.6 shows how the BitTorrent Monitoring system works. The system is divided into two modules, the torrent file searcher (steps 1 and 2 in Figure 2.6) and the torrent file analyzer (steps 3 through 5) [CCM+07].

2.2.3.1 Torrent File Searching. To gather torrents for download and analysis, the investigator seeds the BTM with a list of websites that are known or suspected to contain torrents of illegal files. BTM then performs a depth first search (DFS) beginning with each seed website on the list and linking to a predetermined number of hyperlinks away from that seed site. At each site it encounters, BTM downloads all the .torrent files contained on the webpage and stores them on the

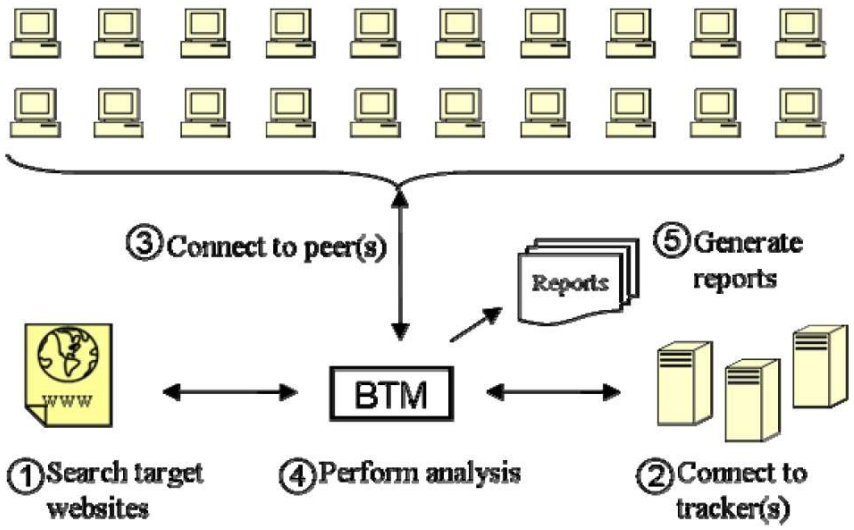

Figure 2.6: The BitTorrent Monitoring System Process [CCM+07]

investigator's computer. These searches can be automatically performed periodically according to any time interval set by the investigator.

2.2.3.2 Torrent File Analyzer. Once a group of torrents is downloaded, the file searcher module hands the list off to the file analyzer module. The analyzer module then compares each .torrent file against a predetermined set of rules that will identify the file as potentially illegal. These rules include such attributes as name, country of origin, creation date, and number of seeders.

If the torrent meets the criteria set down by the rules engine, the module then contacts each peer contained in the file and attempts to download the suspected illegal file. The results of each attempt are recorded by the module, and are then compiled into a series of reports that are output to the investigator. At the completion of the process, the investigator has information on what torrents were downloaded, which of them were determined by the rules engine as potentially illegal, and tracking information on the peers who provided BTM with the files.

2.2.3.3 Shortcomings of the BTM System. BTM has the potential to become a powerful law enforcement tool in combating illegal file sharing. However,

there are two main drawbacks to the system. First, due to the sheer number of torrent files that are available on most torrent websites, the BTM system currently suffers from a very slow processing time. As the number of sublevels covered by the DFS increases, the number of total torrent files to be analyzed increases exponentially, leading to a drastically reduced total processing time.

Second, the BTM system cannot run in real time. As stated above, BTM relies on the torrent file for generating its list of peers to connect to. However, the actual peers that are serving the file for download are constantly changing, with peers being added and deleted continuously. The peer list downloaded from the tracker server only provides a snapshot of active peers, which quickly becomes outdated, limiting its use by BTM. In addition, because the BTM system downloads contraband files to track illicit file sharers, the investigator must ensure that he has the proper legal authority to download the illicit files.

2.2.4 The CopyRouter Peer-to-Peer Tracking System. In October 2008, MSNBC reported that an Australian company, Brilliant Digital Entertainment Ltd., was marketing a new Internet monitoring tool known as CopyRouter [DS08a]. As described by Brilliant Digital Entertainment, the CopyRouter system inspects every packet entering or leaving a target network, looks for peer-to-peer search results that reference files that are on a known contraband list (such as child pornography, the stated primary application of the system), and replaces the illicit file reference with one that leads the user to a law enforcement server instead [DS08b].

Figure 2.7 shows how the CopyRouter system works [DS08b]. When a user inside a network being monitored by CopyRouter attempts to download a file:

1. The user's peer-to-peer client sends a file request to other peer-to-peer clients outside the network on the Internet. CopyRouter does not inspect or modify this search request.

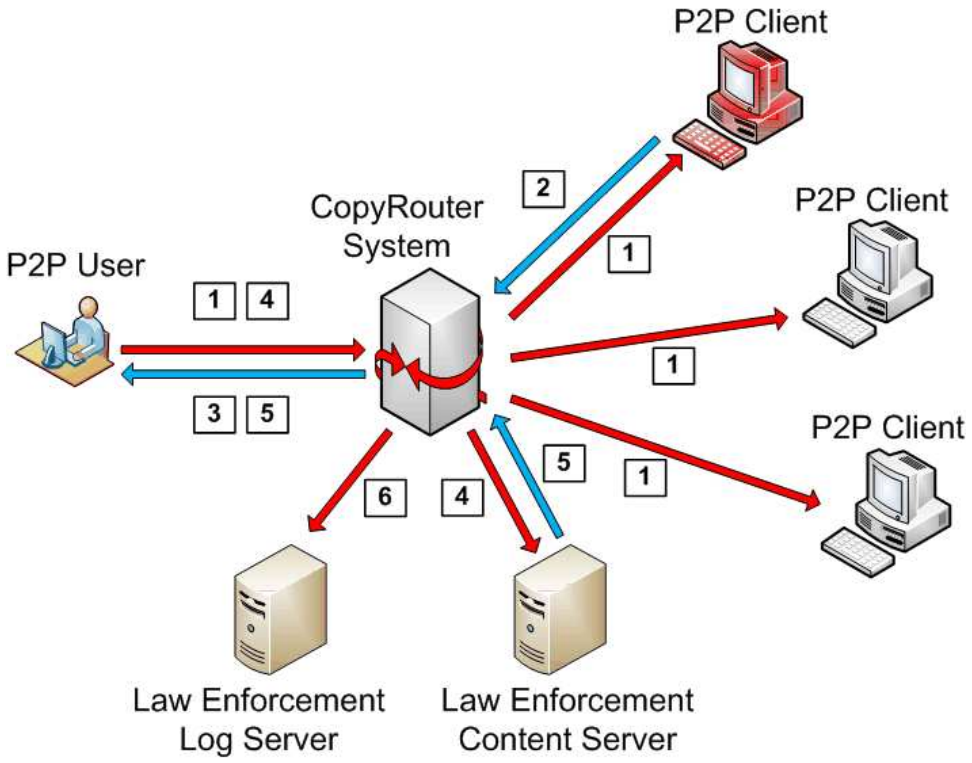

Figure 2.7: How the CopyRouter System Works

2. When an outside peer-to-peer client finds the file, it sends a response to the user indicating that it has the file. This response is intercepted by CopyRouter system, and the hash value of the file described in the response is compared to an internal list of hashes called the Global File Registry (GFR). If the hash value does not match an entry in the GFR, it is passed along without modification.

3. If the file hash of the response matches an entry in the GFR, CopyRouter changes the hash value and the IP address of the client where the file can be found. The IP address is changed to the address of the law enforcement content server and the hash value is changed to match an image file on the law enforcement server notifying the user of the infringement. The modified response is then forwarded to the user.

4. When the user attempts to download the illicit file, he is redirected to the law enforcement content server.

5. The law enforcement content server then sends the user the infringement notice instead of the illicit file.
6. A copy of the original peer-to-peer client response message as well as the modified response message are recorded on a law enforcement log server for further analysis.

Some descriptions of the CopyRouter system claim that it can also hash images, audio files, and video files in real time and compare those files against the GFR [DS08a]. If true, the system may also be used to scan email attachments, HTTP sessions, and FTP transfers in real time.

2.2.4.1 Shortcomings of the CopyRouter System. While the CopyRouter system seems like an effective child pornography tracking system, there are several lingering questions surrounding its implementation. First, CopyRouter is a proprietary system, and to date, Brilliant Digital Entertainment has yet to release any specifications or experimental data on the system's speed, effectiveness, or ability to process all packets at full network speed.

Second, while seemingly effective for peer-to-peer systems where only one uploader is involved, such as Gnutella, the system's ability to monitor distributed peer-to-peer systems such as BitTorrent is questionable. As discussed in Chapter 3, as a BitTorrent client downloads pieces of the file from peers, each piece is hashed and compared against the .torrent file. If the hashes do not match, which is always the case when the user downloads from the law enforcement content server, the piece is simply discarded.

Finally, CopyRouter is an active detection and tracking system, meaning that each packet entering or leaving the network is read and possibly modified before being allowed to continue through the gateway. One consequence of this is that the system's presence can theoretically be detected by users with enough knowledge of how the system works. These users can then modify their behavior and simply not use the monitored network to share illicit information.

2.3 Traditional Methods for Classifying Network Traffic

One major component of this research is analyzing network traffic in real time in order to identify packets used by peer-to-peer protocols. Currently, there are three accepted traditional methods of performing classification analysis on network traffic. Port matching involves comparing the traffic's source and destination port against a list of known port numbers in an attempt to match the traffic to a protocol. Payload analysis analyzes the first portion of a packet's payload and attempts to match these byte strings to a specific protocol. Finally, transport level identification uses network flow characteristics to identify the protocol being used without payload analysis or port matching, where a "network flow" is defined as a series of packets flowing on a network that all contain the same 5-tuple of (source IP address, source port number, destination IP address, destination port number, protocol used) [CGD07].

2.3.1 Socket-Layer / Port Matching. The simplest and earliest used method of identifying protocols used by Internet traffic is port matching. In port matching, the network traffic analyzer extracts the source and destination transport layer port numbers from each packet, then compares the port numbers against a table containing a list of peer-to-peer protocols and their corresponding ports. Table 2.1 contains the TCP port numbers for commonly used peer-to-peer protocols.

Table 2.1: Common TCP Port Numbers Used by P2P Applications [Spe08]

P2P Protocol	TCP Port Numbers
eDonkey2000	4661-4665
FastTrack / KaZaA	1214
BitTorrent	6881-6889
Gnutella	6346-6347
Napster	6699
MP2P	41170 UDP
Direct Connect	411-412

2.3.1.1 Advantages of Port Matching. The main advantage of classifying Internet traffic by port matching is its simplicity and speed. In port matching, the only piece of information used to classify the traffic is the port number from the TCP header, which makes packet analysis extremely simple. The simplicity of extracting the port number, combined with the ease of comparing the number against a list of protocols, results in very fast analysis of packets.

2.3.1.2 Drawbacks of Port Matching. In spite of these advantages, port matching is no longer used as an effective Internet classification method. First, most peer-to-peer protocols do not register their port numbers with the Internet Assigned Numbers Authority (IANA). As a result, the IANA cannot accurately track the port numbers used by these applications and make them available to classification software developers.

Second, some peer-to-peer protocols use the same port numbers as other, more well known, applications such as HTTP on Port 80 [Gon05]. This dual-use of well known ports confuse classification software, resulting in an increased number of false negatives when attempting to detect peer-to-peer traffic.

Finally, most peer-to-peer applications allow their users to manually set the port number used by the client. In addition, if the port is not manually specified, newer peer-to-peer clients will instead assign a random port number. In either case, attempting to compare these non-standard port numbers against a static list is futile at best [Gon05]. This weakness is reinforced by a research study in 2005 by Madhukar and Williamson that determined up to 70% of Internet traffic is unclassifiable by straight port matching [MW06].

2.3.2 Application-Layer / Payload Analysis. Another traditional method of identifying Internet traffic by protocol is analyzing the payloads of the packet stream and searching for telltale byte streams that correspond to a specific protocol. In their research, Sen, Spatscheck, and Wang develop a system to quickly identify

packet streams using peer-to-peer protocols by analyzing the first few bytes of the packet payloads [SSW04]. Listed below are identifying features they used to detect five major peer-to-peer protocols by payload.

2.3.2.1 The Gnutella Protocol. The Gnutella protocol uses a series of HTTP-like commands to request files and reply to file requests. When attempting to establish a connection, the requester sends the following as the payload of a TCP packet:

```
GNUTELLA CONNECT/<protocol version string>\n\n
```

The affirmative response to the request for a connection uses the following format:

```
GNUTELLA OK\n\n
```

Based on these observations, Sen, et. al. recommend searching for the string `GNUTELLA` immediately following the TCP/IP header to identify Gnutella traffic.

2.3.2.2 The eDonkey Protocol. The eDonkey protocol uses a common header for both file requests and responses. The header is composed of 5 bytes, where the first byte is a marker byte of value 0xe3, and the next 4 bytes are a number that represents the size of the entire packet minus the TCP/IP headers and the 5-byte header. Sen, et. al. recommend using these 5 bytes to identify eDonkey traffic.

2.3.2.3 The Direct Connect Protocol. The Direct Connect protocol is composed of a series of commands that are used by both the file requester and file provider, and always follows the following format:

```
$command_type message1 message2 message 3 ... |
```

All of the commands in Direct Connect begin with the $ character and end with the | character. Some examples of messages sent within these commands are `Hello, ConnectToMe, Search,` and `Quit`. Sen, et. al. recommend using these beginning and ending characters as a preliminary filter for Direct Connect traffic.

2.3.2.4 The KaZaA Protocol. The KaZaA protocol, like the Gnutella protocol, uses HTTP-like commands to request files. The HTTP format for a file request contains the following fields:

```
GET /.files HTTP/1.1\r\n
Host:  <IP address>/<port>\r\n
UserAgent:  KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
```

The receiver then sends the following back to the requester:

```
HTTP/1.1 200 OK\r\n
Content-Length: \r\n
Server: KazaaClient \r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: \r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
Content Type: \r\n
```

To identify KaZaA traffic, Sen, et. al. recommend searching for the string `GET` or `HTTP` immediately following the TCP/IP header, and then searching the remainder of the packet for a field containing the string `X-Kazaa`.

2.3.2.5 The BitTorrent Protocol. Finally, in the BitTorrent protocol, clients establish a connection with each other through a unique handshake followed by a continuous stream of data. For the connection handshake, the requester and provider both send packets that share a common header of the form:

```
0x13 <The 19 byte string: "BitTorrent Protocol">
```

This header is simply one byte that represents the decimal number 19 in hexadecimal followed by a 19-byte string that represents the phrase "BitTorrent Protocol". Sen, et. al. recommend searching for this 20-byte string immediately following the TCP/IP header to identify BitTorrent traffic.

2.3.2.6 Advantages of Payload Analysis. There are several advantages to using payload analysis for detecting peer-to-peer traffic. First, due to the uniqueness of the payload message headers, payload analysis is very accurate. Second, because accurate identification only requires the analysis of the first 5 to 20 bytes of the payload, identification can be performed very quickly. In their research, Sen, Spatscheck, and Wang were able to identify peer-to-peer connections in less than 10 packets with less than 5% false positive and false negative rates [SSW04].

2.3.2.7 Drawbacks of Payload Analysis. There are also some drawbacks to using payload-based identification of traffic. First, some protocols such as Gnutella are completely open source, which enables third parties to easily modify the protocol for their specific client software. These differing implementations can cause false negatives if the header byte format is changed from the standard implementation of the protocol. Conversely, other protocols such as KaZaA are proprietary, which results in a homogeneous implementation by clients but also limits, or eliminates, the availability of the protocol specification to construct the detection rules from.

Second, payload analysis is vulnerable to obfuscation through the use of byte padding, which adds random characters to the beginning of the payload in order

to move the message headers deeper into the payload, thereby avoiding detection. Byte padding is discussed further in Section 2.5.1. In addition, if the payloads are encrypted, payload analysis is not feasible. Payload encryption is discussed further in Section 2.5.2.

2.3.3 Transport-Layer / Statistical Identification and Analysis. In their research, Karagiannis, Broido, Faloutsos, and Claffy introduce a method of identifying peer-to-peer traffic by analyzing the connection patterns of various peer-to-peer networks [KBFC04]. Introduced in 2004, this method quickly gained acceptance as a new standard for identifying peer-to-peer traffic within large networks. By observing the connection patterns of the source and destination IP addresses during a peer-to-peer file exchange, the researchers were able to design two heuristics that can be used to identify peer-to-peer traffic solely from these connection patterns.

2.3.3.1 Heuristic Using TCP/UDP IP Pairs. The first heuristic searches for IP address pairs that exchange information using both TCP and UDP. The researchers found that many peer-to-peer protocols such as Gnutella, eDonkey, FastTrack, and Direct Connect use UDP for exchanging control and management information and TCP for the actual file transfers. However, other common protocols such as DNS, NETBIOS, and IRC use the same combination of TCP and UDP. Since these applications use well defined port numbers, the heuristic simply ignores traffic from these ports to reduce the number of false positives.

By looking for IP address pairs that exchange information using both TCP and UDP, and ignoring traffic using port numbers that correspond to non-peer-to-peer protocols, the heuristic is able to identify peer-to-peer traffic flows with great accuracy when combined with the second heuristic described below.

2.3.3.2 Heuristic Using (IP, port) Pairs. The other heuristic designed by the researchers is based on evaluating the pattern of connections between source and destination (IP address, port number) pairs. As discussed previously, in a decen-

tralized peer-to-peer network, each client possesses a host cache which contains the addresses of several superpeers, which allows an initial connection into the network. When the superpeer receives the connection request from the client, it forwards the client's (IP, port) pair to other peers within the network, who then establish connections with the client.

Because the client's listening port number is selected randomly, the superpeer must forward both the IP address and the listening port number to the peers for them to facilitate a direct connection. So, each time a peer connects to the client, a network flow is created with the client's (IP, port) pair as one end of the connection. The researcher's heuristic takes advantage of this fact by searching network traffic for a series of connection to a specific (IP, port) pair from many other unique (IP, port) pairs.

By looking for network flows that involve many sources connecting to a single destination (IP, port) pair, this heuristic, when combined with the TCP/UDP pair heuristic, is able to identify peer-to-peer traffic flow with over 95% accuracy [KBFC04].

2.3.3.3 Advantages of Transport Layer Identification of Peer-to-Peer Traffic. There are several advantages to using transport layer identification to detect peer-to-peer networking flows. First, because the two heuristics only depend on a 5-tuple of (source IP, source port, destination IP, destination port, transport layer protocol), the identification algorithm can analyze network traffic very quickly, allowing for real time analysis of network backbone traffic. Second, because the heuristics are based on the connection patterns only, the algorithm can detect peer-to-peer traffic using modified or previously unknown protocols.

2.3.3.4 Drawbacks to Transport Layer Identification. There are also several drawbacks to this method of identifying peer-to-peer traffic flows. First, and most significantly, this method cannot identify the specific peer-to-peer protocol being used, nor can it identify what is being sent. Second, while this method is able to iden-

tify greater than 95% of all peer-to-peer communication, the false positive rate ranges from 8% to 12% [KBFC04]. This high rate may limit its usefulness to Internet service providers seeking to block or shape peer-to-peer traffic, as they would inadvertently block a sizeable percentage of legitimate traffic as well.

2.4 Non-Traditional Methods for Classifying Network Traffic

In the field of network traffic classification, research is continually being performed in the search for newer, more effective methods of classifying Internet traffic by protocol. Discussed below are a sampling of non-traditional methods for identifying network traffic protocols that use Markovian signatures, the discreteness of remote hosts, clustering algorithms, and flow records.

2.4.1 Markovian Signature-Based Classification. In their research, Dahmouni, Vaton, and Rosse describe a real-time method of classifying network traffic by application protocol using a Markovian signature-based approach [DVR07]. They constructed a two part process that enumerates Markov chains that represent the packet flows contained in each target application protocol, and then compares incoming traffic against the Markov chains to find a match.

In the first part of the classification algorithm, a Markov chain is constructed for each application protocol for which the user wishes to identify traffic. Then, based on a training dataset which contains sample network flows using the protocols, the state transition probabilities for each application are created and stored. These state transition probabilities represent the probabilities of transitioning from one packet format used by the protocol to another packet format (e.g., a `GNUTELLA CONNECT` message from a host followed by a `GET /get` command message from the host once the connection is confirmed).

In the second part of the algorithm, the Maximum Likelihood theory and the Neyman-Pearson theory are applied to the incoming network traffic and the Markov chains to decide if the traffic matches an application's Markov model [DVR07]. The

algorithm computes the likelihood value for each Markov model against the unknown traffic flow, and the Markov model with the highest computed likelihood value is assigned to the flow. Although Dahmouni, et. al. only focused on the HTTP, HTTPS, and telnet protocols in their research, the method can be expanded to include peer-to-peer networking applications [DVR07].

2.4.2 Discreteness of Remote Hosts. In their research, Cheng, Gong, and Ding present a new methodology for identifying BitTorrent-like traffic using what they term the "Discreteness of Remote Hosts (RHD)" [CGD07]. RHD takes the total number of concurrent network traffic flows into a specific host (IP address, port number) pair, and measures how much the flows are spread out among differing subnetworks and remote hosts. A high RHD indicates that the concurrent network flows into a host are spread among many different remote networks, indicating that the flows are most likely part of a BitTorrent file transfer. A low RHD indicates that the concurrent network flows are concentrated among few remote networks, indicating a client-server interaction such as an HTTP or an FTP session.

As previously stated in Section 2.3.3, the BitTorrent protocol relies on multiple peers all connecting to the same host (IP address, port number) in order to download files. The RHD algorithm takes advantage of this fact by giving a numerical measurement to how discrete the concurrent network flows are; if the RHD measurement passes a predefined threshold, the flow is labeled as a BitTorrent-like peer-to-peer session. The RHD method of classifying BitTorrent-like traffic returned an accuracy of better than 90%, on average [CGD07].

2.4.3 Clustering Algorithms. In their research, Erman, Arlitt, and Mahanti develop a classification process that relies on using a clustering algorithm to group network flows by protocol used [EAM06]. The clustering algorithm computes the Euclidian distance between pairs of network connections, where each network connection is defined as a packet flow between two hosts. Each packet flow consists of the number

of packets in the flow, the mean packet size, the mean payload size, the total number of bytes transferred, and the mean inter-arrival time of the packets.

The classification process is composed of two parts. First, an unsupervised clustering algorithm such as K-Means (which partitions objects into K subsets based on squared-error calculation for each object) or DBSCAN (which assigns objects to clusters based on the density of other objects in its immediate vicinity) is applied to a set of test data in order to build and accurately label a set of clusters, which becomes the classification model. Second, the model is used to build a classifier tool, which the researchers have not yet constructed, that can label network traffic either offline, using packet capture files, or in real time on a live network.

The researchers note that this classification technique is still in the developmental stage. However, even with this limitation, they are still able to correctly classify Internet traffic with an accuracy of approximately 80% [EAM06].

2.4.4 Flow Records. In their research, Karagiannis, Papagiannaki, and Faloutsos outline a novel approach to the traffic classification problem that they term BLINd Classification (BLINC) [KPF05]. This classification framework attempts to characterize network flows on three levels: The social level, the functional level, and the application level. By combining data gleaned from each of these levels, an accurate identification of the network flow can be made.

2.4.4.1 The Social Level. At the social level, BLINC measures the popularity of each host in the network by measuring the number of other hosts it communicates with. As shown in Figure 2.8, the BLINC system analyzes the source and destination IP addresses of each packet flowing through the network, and groups them together to form bipartite clusters of hosts. By analyzing these graphs using various cross-association algorithms, BLINC can determine the diversity of the connections made to and from each host, and then identify the popularity of each host and frequency of communications between hosts.

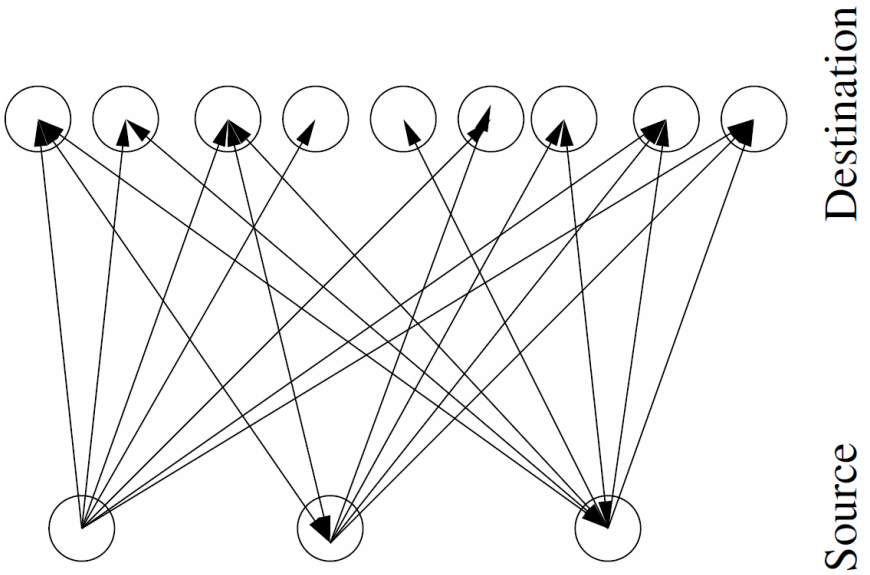

Figure 2.8: Example of a Bipartite Cluster of Network Hosts [KPF05]

2.4.4.2 The Functional Level. At the functional level, BLINC uses connection information to determine the function of each host on the network, such as a server, consumer, or collaborator. For example, a host that receives many incoming connections on one listening port is most likely a server of information to various consumers. For this analysis, BLINC looks at the source and destination IP addresses, as well as the source port number for each packet.

2.4.4.3 The Application Level. At the application level, BLINC looks at the transport layer interactions between hosts in an attempt to determine what applications are being used for the packet flows. After examining the source and destination IP addresses, as well as the source and destination port numbers, BLINC constructs a graphlet, which is a pattern of behavior for a given network flow. Figure 2.9 shows examples of these graphlets for some common applications found on the Internet. In each graphlet, the relationship between the source and destination IP addresses, shown in the first two columns, and their relationship to the source and destination ports, shown in the last two columns, are expressed by lines connecting the various nodes. By constructing a library of these graphlets, each of which has a

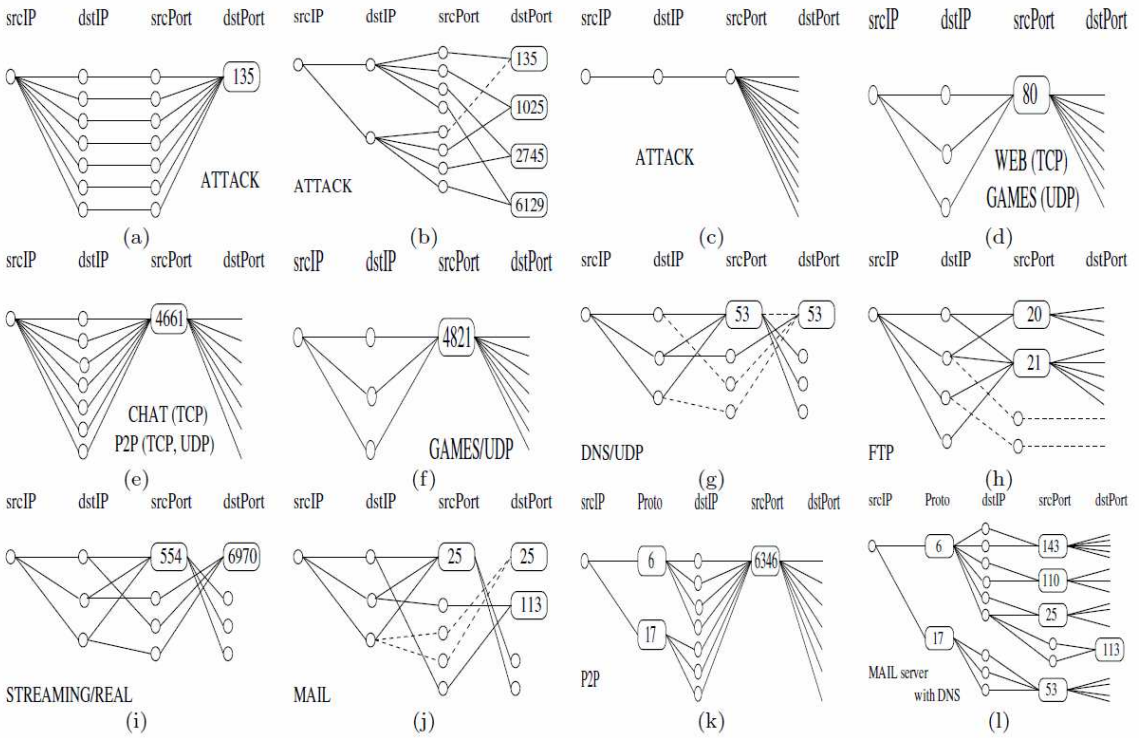

Figure 2.9: Transport Layer Interactions for Various Applications [KPF05]

unique and easily recognizable shape and connection pattern, BLINC can then compare an unknown traffic flow against the library and attempt to match it to a known application.

Through the use of a series of heuristics based on these three levels, the researchers were able to classify 80-90% of network traffic with an accuracy of greater than 95% [KPF05].

2.5 Obfuscation of Peer-to-Peer Traffic

As previously discussed in Section 2.1.3, peer-to-peer networks account for a majority of Internet traffic. In order to combat this trend, and to preserve their bandwidth, some Internet service providers (ISPs) are taking active measures to throttle peer-to-peer traffic on their networks. For example, Comcast has developed a method of throttling peer-to-peer traffic, whereby when the ISP's traffic classifiers detect an extremely active peer-to-peer connection, it transmits `RESET` messages to the sender

and receiver to break the TCP connection [Sve07]. In order to combat traffic throttling by ISPs, peer-to-peer clients have begun using obfuscation techniques to defeat the traffic classifiers. Outlined below are three methods of traffic obfuscation: Byte padding, Encryption, and Tunneling.

2.5.1 Byte Padding. One method of preventing a payload-based traffic classifier from accurately identifying peer-to-peer traffic is to use byte padding. In byte padding, a series of random characters is appended to the beginning of every packet sent by the peer-to-peer client. Since payload analyzers usually look at the first 16 to 40 bytes of payload to determine the protocol used, byte padded packets will look like encrypted traffic and will be labeled as "unknown."

Though this method of obfuscation is effective against primitive payload-based classifiers, it suffers from several shortcomings. First, the messages identifying the packet as belonging to a peer-to-peer protocol are still visible; they just appear later in the payload. By modifying the payload-based packet analyzer to search the entire payload of each packet, this obfuscation technique can be defeated. Second, this obfuscation technique has no effect on network flow-based classification methods that do not analyze the payload at all in order to identify the application being used.

2.5.2 Encryption. A more sophisticated method of obfuscation is to encrypt the payloads of the packets being exchanged on the peer-to-peer network. Encrypting the payloads renders payload-based classification useless, since there is no longer any useful information available. For example, the eMule peer-to-peer client (which uses the eDonkey peer-to-peer protocol) allows the user to enable "protocol obfuscation", in which users exchange keys using public key cipher, then exchange data using RC4 bit stream encryption [Pro08b].

As with byte padding, encryption has no effect on transport-layer traffic identifiers, since the TCP/IP headers are still intact (only the payload is obfuscated), and the payload is not analyzed in these methods.

2.5.3 Tunneling. Currently, the best method of obfuscating peer-to-peer network communication is to use a tunneling mechanism such as secure shell (SSH) or a virtual private network (VPN). In SSH and VPN, the packets containing the peer-to-peer protocol are themselves encrypted and encapsulated by the SSH/VPN protocol, then sent on the network as an SSH or VPN packet. This restricts the amount of useful data about the underlying peer-to-peer protocol to the source and destination IP addresses, approximate packet size, and timing. Using an SSH or VPN tunnel will defeat almost any classification method, most of which will simply identify the peer-to-peer traffic as an encrypted tunneling protocol.

However, there has been research done on identifying the underlying traffic of an SSH or VPN connection. In their research, Gebski, Penev, and Wong describe a method of identifying the underlying applications within tunneling traffic [GPW06]. By constructing bipartite graphs of request and response pairs, then combining these graphs with timing and size information, the researchers were able to identify BitTorrent traffic with 90% accuracy.

In their research, Wright, Monrose, and Masson achieve the same results while using the same network information, namely packet size, timing, and direction of flow [WMM06]. By using a combination of k-Nearest Neighbors and hidden Markov model classifiers, the researchers were able to achieve accurate detection rates of 80-90% for common protocols such as HTTP and telnet. In addition, they were able to actually track the number of flows within an encrypted tunnel if they all use a single protocol. Note that both of these methods can only confirm the existence of peer-to-peer traffic; it cannot determine what is being downloaded.

2.6 Summary

This chapter presents background information on the evolution of peer-to-peer networking and the major types of peer-to-peer protocols that are currently being employed. Several current methods of identifying and tracking illicit file sharers are discussed and analyzed. Current research in the areas of traditional and non-traditional

network traffic classification are studied. Finally, the most common techniques for obfuscating and encrypting peer-to-peer traffic are discussed. Based on the information provided in this chapter, payload analysis is selected as the traffic classification method for the TRAPP system, and the issues of obfuscation and encryption are not considered in this research due to the complexity of the problem.

III. The BitTorrent Peer-to-Peer Networking Protocol

This chapter presents an overview of the BitTorrent peer-to-peer protocol, which will be used extensively for this research. To provide a foundation for understanding the BitTorrent protocol, Section 3.1 gives an overview of how the protocol functions to distribute files, Section 3.2 discusses specific terminology necessary for an understanding of the protocol, Section 3.3 provides a primer on how to encode information using BitTorrent's bencoding system, and Section 3.4 provides a general discussion of how the SHA-1 hash algorithm works. Once this foundation is laid, Section 3.5 delves into the inner workings of the .torrent file, Section 3.6 describes the BitTorrent tracker protocol which is used to request files, and Section 3.7 describes the BitTorrent peer wire protocol which is used for the actual transfer of files.

3.1 Overview of How BitTorrent Works

One of the greatest drawbacks to Gnutella and other distributed peer-to-peer protocols discussed in Section 2.1.2 is the fact that most users can download files at much greater speeds than they can upload them [Rea05]. Because Gnutella-like protocols involve one peer uploading a file to another peer that downloads it, the total transfer speed is capped by the upload speed, which can be many times slower than the download speed.

In 2001, Bram Cohen developed a new protocol called BitTorrent that was designed to take advantage of high download speeds while mitigating the effect of slow upload rates [Coh03]. The BitTorrent protocol differs from other distributed peer-to-peer protocols in that it allows downloaders to download pieces of files from tens or hundreds of other users simultaneously. To further speed up downloads, every user that downloads pieces of files also uploads those pieces he already possesses. By aggregating the slower upload speeds of hundreds of peers, the protocol can achieve very high download rates.

3.1.1 The Purpose of BitTorrent. BitTorrent is designed to give individuals and businesses the power to distribute enormous amounts of data using a reduced amount of bandwidth, which places less strain on content provider servers and reduces the costs associated with the bandwidth. For businesses, this distributed system also provides data redundancy in that there is now no single point of failure for loss of data. For individuals, BitTorrent provides the fastest possible speeds due to its ability to download data from many other peers simultaneously [Bas08].

In fact, according to BitTorrent's author, Bram Cohen [Coh08]:

> Cooperative distribution can grow almost without limit, because each new participant brings not only demand, but also supply. Instead of a vicious cycle, popularity creates a virtuous circle. And because each new participant brings new resources to the distribution, you get limitless scalability for a nearly fixed cost.

3.1.2 Downloading a File Using BitTorrent. Figure 3.1 shows how the BitTorrent protocol is used to download a file. As shown in the figure, when downloading a file, the following occurs:

1. The user searches the Internet for a web site that contains the .torrent file for the file he wishes to download.
2. The user then downloads the .torrent file and uses a BitTorrent client to open the file.
3. The client uses the metadata contained in the .torrent to contact the tracker and submit a peer list request using the BitTorrent tracker protocol.
4. The tracker sends the client a list of all peers currently seeding the file referenced in the .torrent file.
5. The client sends handshake messages to the peers on the list sent to it by the tracker, and joins the swarm. If the remote peer has the file, it will respond.
6. The client then proceeds to download pieces of the file from each seeder that responds to the handshake message using the BitTorrent peer wire protocol. As

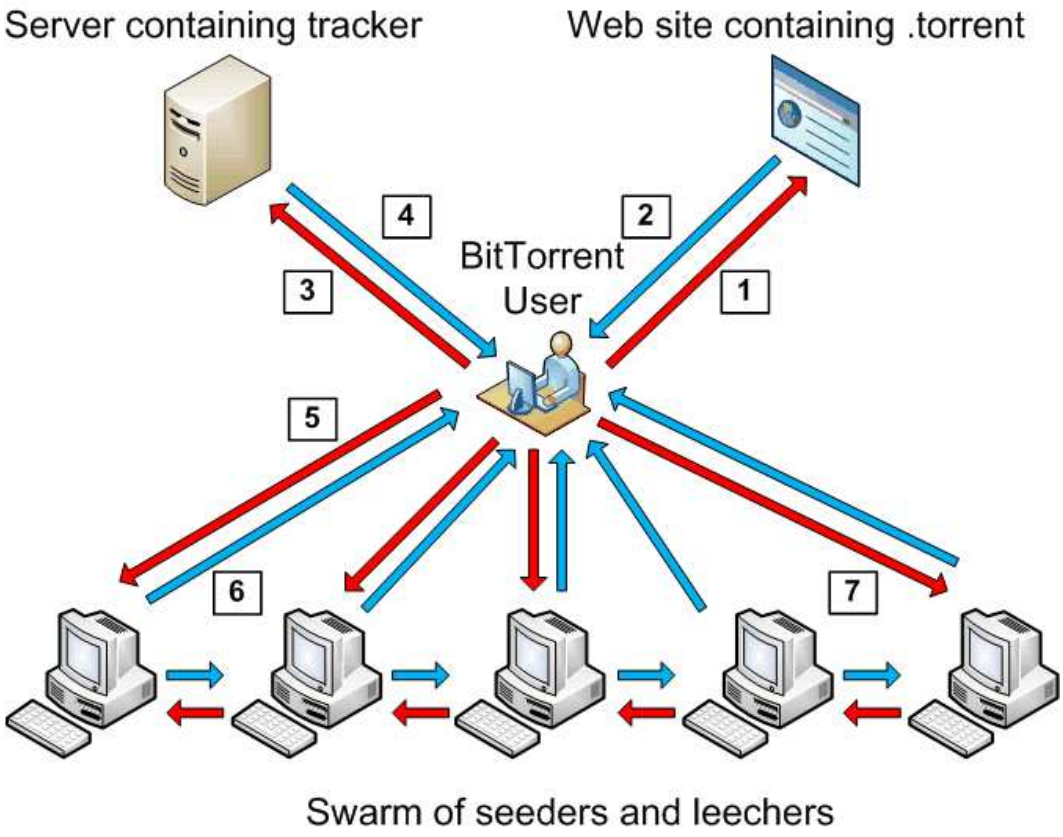

Figure 3.1: How the BitTorrent Protocol Works

the client completes the download of a piece, it immediately provides the piece to other peers in the swarm for them to download.

7. Once the entire file is downloaded, the client continues to act as a seeder for the file until the user disconnects the client from the swarm.

3.1.3 An Example of the Power of the Swarm. In *Wired* magazine, Clive Thompson provides a startling account of the power of the BitTorrent peer-to-peer model [Tho05]. A graduate student in computer science at Stanford named Gary Lerhaupt was intrigued by the documentary *Outfoxed*, and decided that he wanted to make it available to more people. Lerhaupt contacted the film's producer and asked for and received permission to post a portion of the documentary on his web site as a torrent. Over the next two months, approximately 1,500 people downloaded the 500-megabyte video, consuming around 750 gigabytes of traffic. However, because of

the unique nature of BitTorrent, Lerhaupt himself only served up 5 gigabytes; the rest of the data was provided by other seeders that had downloaded the file from Lerhaupt and then served up portions themselves. In essence, BitTorrent allowed Lerhaupt to effectively increase his data output by a factor of 150 by using the power of the swarm.

3.2 Terminology

The BitTorrent protocol uses a specific lexicon to describe different aspects of the network's components and functions. Outlined below are some of the more common terms used when describing the BitTorrent protocol, and most are used later in this section [Dir07].

3.2.1 Torrent. The term *torrent* has two possible meanings, depending on context. In one case, the term torrent refers to the .torrent file that contains the metadata necessary to download a specific file or group of files. In the other case, the term refers to the actual file or group of files being shared.

3.2.2 Peers and Swarms. In BitTorrent, a *peer* is any computer running a BitTorrent client program that is connected to the Internet and is available for uploading or downloading information. Any collection of peers that is actively sharing a particular torrent is called a *swarm*.

3.2.3 Seeders and Downloaders. As with any other peer-to-peer protocol, BitTorrent has a number of computers uploading or downloading information to and from each other. A *seeder* is a peer that possesses a complete copy of a file and is actively making it available to other peers for download. A *downloader*, on the other hand, is a peer that does not have a complete copy of the torrent and is actively retrieving it from other peers. As stated previously, any peer that downloads a torrent is also serving up those pieces already downloaded to other peers. So, in the BitTorrent protocol, all downloaders are also uploaders (at least until the file is completely

download, at which time the downloader can choose whether or not to be a seeder for the torrent).

3.2.4 Leeches and Lurkers. The success of the BitTorrent protocol depends on peers offering files for seeding once they are downloaded. A *leech* is a peer that does not make files available for seeding once they are downloaded, does not add any new content for download, and may artificially reduce its upload speed to limit the amount of data uploaded. A *lurker* is similar to a leecher in that the peer does not offer any new content for download. However, a lurker, unlike a leecher, becomes a seeder for any files downloaded.

3.2.5 Trackers. A *tracker* is a server whose purpose is to keep track of what seeders and peers are available in a particular swarm. The tracker usually also contains a web server containing .torrent files that are available for download. The tracker only provides coordination; it is in no way involved in actual file transfer and does not contain copies of files to share.

3.2.6 Blocks versus Pieces. When a file is made available for downloading, it is divided into *pieces*, which are equal sized portions of the file (except for the last piece), and are almost always a power of 2 in length, measured in bytes. In the latest version of the BitTorrent protocol, the piece length defaults to 2^{18} bytes = 256 kilobytes [Coh08]. Each piece is hashed using SHA-1 and the resulting 160-bit hash is placed in the .torrent file. A *block* is a portion of a piece that is requested by the downloader; when all blocks in a piece are retrieved, they are combined into a piece, error checked, then made available for download to other peers.

3.3 Bencoding

Bencoding (pronounced "Bee Encoding") is the method that BitTorrent uses to encode and organize data within .torrent files [Coh08] [Bit08]. It provides support

for encoding integers, byte strings, lists, and dictionaries. Outlined below are the encoding formats and how they are created.

3.3.1 Encoding Integers. To encode an integer, use the following format:

```
i<integer encoded in base 10 ASCII>e
```

Examples: `i786e  i-8754e`

The `i` and `e` are the beginning and ending delimiters for the number. For negative numbers, insert a "-" sign before the number. The use of leading zeros (e.g., `i00045e`) is prohibited.

3.3.2 Encoding Strings. To encode a byte string, use the following format:

```
<string length in base 10 ASCII>:<byte string>
```

Examples: `6:pieces  4:name`

In this case, there are no beginning or ending delimiters. The contents of the byte string can contain either a string of ASCII characters or, in the case of SHA-1 hashes, pure binary data. Thus, the use of a hex editor is essential to read the contents of a bencoded byte string.

3.3.3 Encoding Lists. To encode a list composed of any other type of bencoded data, use the following format:

```
l<bencoded data>e
```

Example: `l6:piecesi256ee`

Note that in the example, the list ends with two `e`'s. The first `e` delimits the integer value and the second `e` delimits the list.

3.3.4 Encoding Dictionaries. To encode a dictionary consisting of elements and keys, use the following format:

```
d<bencoded key><bencoded element><bencoded key><bencoded element>e
```

Example: `d6:piecesi64e6:lengthi16256ee` represents `pieces => 64` and `length => 16256`.

In the case of dictionaries, the keys must be bencoded strings, and the values may be any bencoded type, including integers, strings, lists, or other dictionaries.

3.4 The SHA-1 Hash

RFC 3174 was created to implement the Secure Hash Algorithm 1 (SHA-1), designed by the United States Government and labeled the Federal Information Processing Standards Publication 180-1 (FIPS 180-1) [RFC01]. The SHA-1 algorithm takes a binary message of less than 2^{64} bits and produces a 160-bit output message labeled a message digest. This message digest can then be used to generate a digital signature, providing authenticity for the original message [FIP93].

3.4.1 Creating a Message Digest. As stated in FIPS 180-1, the message digest is created by applying the following series of steps to the original message:

1. Pad the message. The message is padded with a 1, a series of 0s, then a 64-bit integer that represents the total length of the message, so that the total message length is a multiple of 512.
2. Divide the message. The message is divided into 512-bit blocks, where each block is composed of 16 words of 32 bits each.
3. Initialize the constants. Before the algorithm is applied, 5 constants, each 32 bits long and labeled A through E, are initialized with values determined by the standard [RFC01]. These constants are combined with the message words using modular arithmetic to produce the message digest.

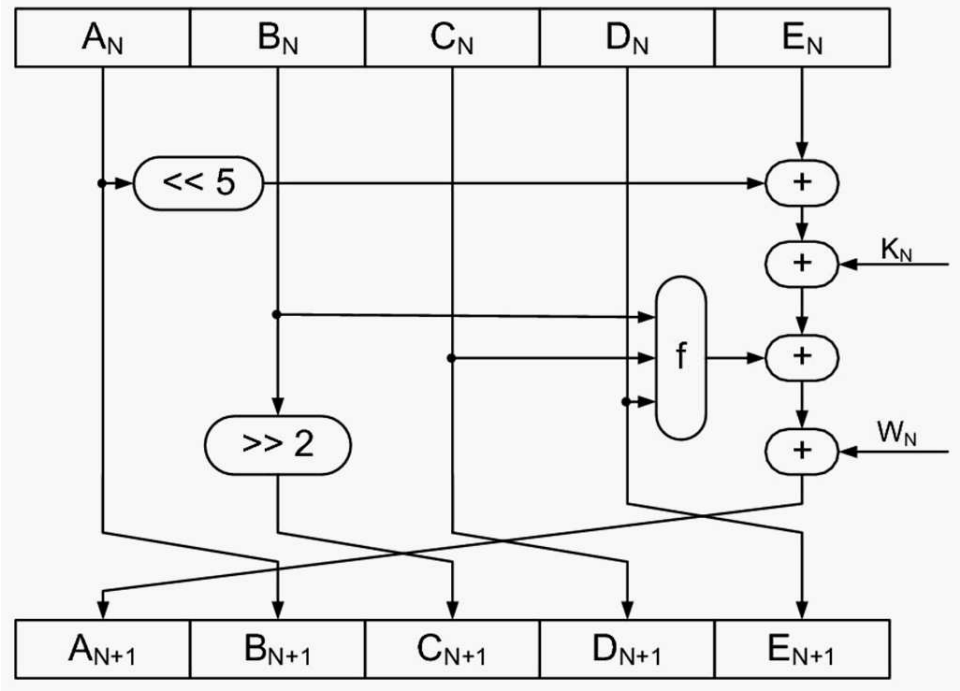

Figure 3.2: How a Round in the SHA-1 Hashing Algorithm Works [MRR08]

4. Apply the algorithm. As shown in Figure 3.2, the 32-bit message words A, B, C, D, and E are taken 5 at a time and run through a series of mathematical operations. In the figure, f is a varying nonlinear function, << denotes a left shift by the indicated number of bits, W is the expanded message word for the round, K is the constant for the round, and + indicates addition modulo 2^{32}. At the end of 80 of these rounds, the 5 words remaining (shown at the bottom of the figure) are concatenated to create the (5 words * 32 bits per word = 160 bits) 160-bit message digest.

3.4.2 Security of the SHA-1 Standard. According to the FIPS 180-1 standard, the SHA-1 hash is considered to be a secure hash because "it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify" [FIP93].

Researchers around the world are currently attempting to construct collisions (two messages that produce the same message digest) using SHA-1. In 2005, a team of researchers in China discovered a method that theoretically can produce a collision using 2^{69} operations. While this number of computations is still unfeasible, it is less than the commonly accepted bound of 2^{80} operations, which is considered to be the minimum number of computations for a system to be secure [MRR08].

3.4.3 Use of the SHA-1 Hash in BitTorrent. The SHA-1 algorithm is used in the BitTorrent protocol for two purposes, error correction and file identification.

In *error correction*, the SHA-1 algorithm is applied to each block of the file to be uploaded via BitTorrent. These message digests are then placed in the .torrent file that the client downloads. Once a block is completely downloaded from the seeders, the client applies the SHA-1 to the downloaded block and compares the message digest to the one in the .torrent file. If the two hashes match, the client assumes that the transfer was error-free.

In *file identification*, the SHA-1 algorithm is applied to the information dictionary contained in the .torrent file. The resulting message digest is labeled as the file identifier, which will uniquely identify the file being offered for download regardless of the file description provided by the tracker. When a downloader requests a file, the client will provide the hash as the file identifier in the GET request. This method prevents two files that have the same description but different contents from being confused with each other when downloading blocks from peers. More information on how to compute the information dictionary hash from the .torrent file is in Section 3.5.3.

3.5 The .torrent File

The .torrent file contains metadata that allows a BitTorrent client to connect to a tracker, obtain a peer list, and download the file requested. The contents of the

.torrent file are outlined below, followed by an example of a .torrent file downloaded from the Internet.

3.5.1 Contents of the .torrent File. All information in the .torrent file is bencoded and contains the following fields [Bit08]:

- `announce`. This is the URL of the tracker from which the peer list will be downloaded.
- `creation date`. This is the creation time of the .torrent file, in standard UNIX epoch format.
- `info`. A bencoded dictionary containing the rest of the items below.
- `length`. This is the length of the file to be downloaded in bytes.
- `name`. The name of the file to be downloaded.
- `piece length`. The length of each piece of the file in bytes.
- `pieces`. A byte stream containing the SHA-1 hashes of each piece concatenated together. The length of this byte stream will always be a multiple of 20.
- `path`. Used to identify the file structure of a .torrent that contains multiple files.

3.5.2 Example of a .torrent File. The following .torrent represents a 2,133,210-byte file, named freeculture.zip which is available from the www.legaltorrents.com website [Les08a]. Note that for this example, the portion of the file containing the <> delimiters (which are not actually contained in the file) and the contents within are the 180 bytes of binary data converted to hexadecimal and represented in ASCII for readability purposes.

```
d8:announce42:http://www.legaltorrents.com:7070/announce13:creation
datei1081312084e4:infod6:lengthi2133210e4:name15:freeculture.zip12:
piece lengthi262144e6:pieces180:<ED89E7384B0D340FA5FFFA19D1A00D88E7
54BE00505C6EDB15D307AA8E463891F4D2530E8DD1390A0476F3973BA169B561031
```

```
ECC4328751A4C0094E1F23149441F4A2C95E2A7C8C1804BBDF379F39DDC4E4356CB
152A7A4B6BC5CC1C9079ABD1E4F7C4225E685CFC63562F81AED6911A0F92C83026D
4D8673CA2233427C06B20C2CED026B7B84932B3C12655DBEF49E0057373AC89D056
2758F391A5F7DEAB52B2DD59AF12E3441B8B5393FF8D76DE6BC1910F84>ee
```

Before analyzing this particular .torrent file, it is helpful to add line breaks and spaces to make the contents more readable:

```
d8:announce  42:http://www.legaltorrents.com:7070/announce
13:creation date  i1081312084e
4:info
d6:length  i2133210e
4:name  15:freeculture.zip
12:piece length  i262144e
6:pieces  180:
<ED 89 E7 38 4B 0D 34 0F A5 FF FA 19 D1 A0 0D 88 E7 54 BE 00
 50 5C 6E DB 15 D3 07 AA 8E 46 38 91 F4 D2 53 0E 8D D1 39 0A
 04 76 F3 97 3B A1 69 B5 61 03 1E CC 43 28 75 1A 4C 00 94 E1
 F2 31 49 44 1F 4A 2C 95 E2 A7 C8 C1 80 4B BD F3 79 F3 9D DC
 4E 43 56 CB 15 2A 7A 4B 6B C5 CC 1C 90 79 AB D1 E4 F7 C4 22
 5E 68 5C FC 63 56 2F 81 AE D6 91 1A 0F 92 C8 30 26 D4 D8 67
 3C A2 23 34 27 C0 6B 20 C2 CE D0 26 B7 B8 49 32 B3 C1 26 55
 DB EF 49 E0 05 73 73 AC 89 D0 56 27 58 F3 91 A5 F7 DE AB 52
 B2 DD 59 AF 12 E3 44 1B 8B 53 93 FF 8D 76 DE 6B C1 91 0F 84>ee
```

Deconstructing this example, the following information can be extracted:

- Location of the tracker: http://www.legaltorrents.com:7070/announce
- Creation date: Wednesday, 7 April 2004 at 04:28:04 UTC

- File length: 2,133,210 bytes
- File name: freeculture.zip
- Piece length: 262,144 bytes or 256 Kilobytes per piece
- Number of pieces: 180 bytes of binary data divided by 20 bytes (160 bits) per piece = 9 pieces total.

3.5.3 Computing the Information Dictionary Hash Value. For later portions of the research, the information dictionary hash value, also known as the file info hash, takes on enormous importance. Recall that when a peer wants to download a file, the BitTorrent client will take the information dictionary portion of the .torrent file, apply the SHA-1 algorithm, and use that 160-bit value as the file identifier. This unique 20-byte identifier is then used in the BitTorrent GET request message to ask the swarm for the file.

However, it is not immediately clear exactly which portion of the .torrent file is processed using the SHA-1 algorithm to generate the hash. After an exhaustive brute force search, the portion of the file used for hashing was discovered to be the byte stream beginning with the string `d6:length` and ending with the first `e` after the hash values of the file pieces.

In the example above, the information dictionary hash value is based on the following byte stream:

```
d6:length  i2133210e
4:name  15:freeculture.zip
12:piece length  i262144e
6:pieces  180:
<ED 89 E7 38 4B 0D 34 0F A5 FF FA 19 D1 A0 0D 88 E7 54 BE 00
 50 5C 6E DB 15 D3 07 AA 8E 46 38 91 F4 D2 53 0E 8D D1 39 0A
 04 76 F3 97 3B A1 69 B5 61 03 1E CC 43 28 75 1A 4C 00 94 E1
 F2 31 49 44 1F 4A 2C 95 E2 A7 C8 C1 80 4B BD F3 79 F3 9D DC
```

```
4E 43 56 CB 15 2A 7A 4B 6B C5 CC 1C 90 79 AB D1 E4 F7 C4 22
5E 68 5C FC 63 56 2F 81 AE D6 91 1A 0F 92 C8 30 26 D4 D8 67
3C A2 23 34 27 C0 6B 20 C2 CE D0 26 B7 B8 49 32 B3 C1 26 55
DB EF 49 E0 05 73 73 AC 89 D0 56 27 58 F3 91 A5 F7 DE AB 52
B2 DD 59 AF 12 E3 44 1B 8B 53 93 FF 8D 76 DE 6B C1 91 0F 84>e
```

Running this portion of the .torrent byte stream through a SHA-1 hash value generator, the following message digest is created:

```
<CA 6A C4 BB D9 71 D3 90 29 35 DB CF C2 D3 EA 25 B4 28 A5 47>
```

3.6 The Tracker Protocol

The first protocol used by BitTorrent is the tracker protocol, which runs on top of HTTP. The protocol consists of two messages: The peer list request made by the client to the tracker that also includes file transfer statistics used by the tracker, and the response message from the tracker that provides the client with a list of peers that are seeding the file of interest to the client.

3.6.1 The File GET Request. To request a peer list from the tracker, the client sends a GET request containing the following fields:

1. `GET /announce`. Header for the GET request.
2. `info_hash`. The 20-byte SHA-1 hash of the information dictionary used to uniquely identify the file to be downloaded. Note that since the GET request is transmitted in ASCII, this value will have to be escaped using the %xx format as described in RFC 1738 [RFC94].
3. `peer_id`. A 20-byte string that uniquely identifies the BitTorrent client requesting the file. This ID is created by the client sending the request.

4. `port`. The port number that the client is listening on.
5. `uploaded`. The total amount of data in bytes uploaded since the first `started` message was sent to this tracker.
6. `downloaded`. The total amount of data in bytes downloaded since the first `started` message was sent to this tracker.
7. `left`. The total amount of data in bytes left to be downloaded.
8. `key`. An optional field containing an ASCII string used by a client to reconnect to the tracker should its IP address change.
9. `event`. This may be one of three values:
   (a) `started`. Sent when the client first connects to the tracker.
   (b) `stopped`. Sent when the client disconnects from the tracker before the download is complete.
   (c) `completed`. Sent when the client completes the file download.
10. `numwant`. An optional field that indicates how many peers the client wants the tracker to provide.
11. `compact`. Indicates whether the client can accept a compressed version of the peer list, which is composed of six bytes per peer (4 bytes for the IP address and 2 bytes for the port number).
12. `no_peer_id`. Indicates whether the client is requesting peer_id numbers for each peer in the list.
13. `Host`. The URL and port number of the tracker the request is being sent to.
14. `User-Agent`. The name and version number of the client being used.
15. `Accept-Encoding`. The type of encoding accepted by the client.

3.6.2 The Tracker Response. Upon receiving the GET request, the tracker, if it has the information requested, will send a response containing the following fields:

1. `HTTP/1.0 200 OK`.
2. `Content-Length, Content-Type`, and `Pragma`. Additional information associated with the HTTP OK message.
3. `8:complete`. The number of peers that have the entire file, a.k.a. seeders.
4. `10:incomplete`. The number of peers that are in the swarm, but do not have the entire file, a.k.a. downloaders.
5. `8:interval`. The interval in seconds that the client should wait between sending requests to the tracker.
6. `5:peers`. A byte string containing the IP addresses and port numbers of all peers in the swarm. If the peers are sent in the compacted format (see the GET request above), this byte string will have a length that is a multiple of six.

3.6.3 Tracker Request and Response Example. The following tracker request and tracker response are taken from the download of a 103,372,284-byte file, named freeculture-audiobook.zip which is available from the www.legaltorrents.com website [Les08b]. Note that for this example, the portion of the file containing the <> delimiters (which are not actually contained in the file) and the contents within are binary data converted to hexadecimal and represented in ASCII for readability purposes.

First, the client sends a GET request to the tracker at address 8.12.35.122:7070, which corresponds to the URL http://www.legaltorrents.com:7070/announce, and is the same URL as the previous example:

```
GET /announce?info_hash=%10%1c%9dc%21%1c%3cW%0f%fb%ad%d4%9cVI%d3%fb
Irs&peer_id=-UT1770-%f3%9f%fd%c7t%b5jLSR%c1%1c&port=43438&uploaded=
0&downloaded=0&left=103372284&key=7F29A0C7&event=started&numwant=20
0&compact=1&no_peer_id=1 HTTP/1.1<0D0A>Host: www.legaltorrents.com:
7070<0D0A>User-Agent: uTorrent/1770<0D0A>Accept-Encoding: gzip<0D0A
0D0A>
```

Again, it is helpful to reorganize the message for readability purposes:

```
GET /announce
?info_hash=%10%1c%9dc%21%1c%3cW%0f%fb%ad%d4%9cVI%d3%fbIrs
&peer_id=-UT1770-%f3%9f%fd%c7t%b5jLSR%c1%1c
&port=43438
&uploaded=0
&downloaded=0
&left=103372284
&key=7F29A0C7
&event=started
&numwant=200
&compact=1
&no_peer_id=1
HTTP/1.1 (Carriage Return/Line Feed)
Host: www.legaltorrents.com:7070 (CR/LF)
User-Agent: uTorrent/1770 (CR/LF)
Accept-Encoding: gzip (CR/LF/CR/LF)
```

Note that the `info_hash` and `peer_id` fields are composed of hexadecimal numbers that have been escaped into ASCII.

After the tracker receives the GET request, it sends the following back to the client:

```
HTTP/1.0 200 OK<0D0A>Content-Length: 69<0D0A>Content-Type: text/pla
in<0D0A>Pragma:no-cache<0D0A0D0A>d8:completei8e10:incompletei1e8:in
tervali1800e5:peers12:<4B8B8403F5DBD0452AC21B9F>e
```

And, after reordering for readability purposes:

```
HTTP/1.0 200 OK (Carriage Return/Line Feed)
Content-Length: 69 (CR/LF)
Content-Type: text/plain (CR/LF)
Pragma:no-cache (CR/LF/CR/LF)
d8:complete i8e
10:incomplete i1e
8:interval i1800e
5:peers 12:<4B 8B 84 03 F5 DB   D0 45 2A C2 1B 9F>e
```

Note that the tracker sends back the addresses of two peers that are actively seeding the file. These addresses and port numbers, in dotted quad notation are:

```
 4B    8B    84   03   F5 DB
 75  . 139 . 132 .  3 : 62939
```

and

```
 D0    45    2A    C2   1B 9F
208 .  69 .  42 . 194 :  7071
```

Now that the client has the addresses of two peers, it can use the peer wire protocol to begin downloading the file.

3.7 The Peer Wire Protocol

The peer wire protocol is the other main protocol used by BitTorrent, and runs on top of TCP. The purpose of this protocol is to enable the exchange of file pieces enumerated in the .torrent files. The peer wire protocol serves two main functions, handshaking and exchanging messages pertaining to the actual transfer of file pieces between peers.

3.7.1 The Handshake. Continuing the previous example, the client sends the following handshake request to each of the two peers in the list provided by the tracker.

```
<13>BitTorrent protocol<0000000000100001101C9D63211C3C570FFBADD49C5
649D3FB4972732D5554313737302DF39FFDC774B56A4C5352C11C>
```

As with the .torrent file, before analyzing this particular request, it is helpful to add line breaks and spaces to make the contents more readable:

```
<13> BitTorrent protocol
<00 00 00 00 00 10 00 01
10 1C 9D 63 21 1C 3C 57 0F FB AD D4 9C 56 49 D3 FB 49 72 73
2D 55 54 31 37 37 30 2D F3 9F FD C7 74 B5 6A 4C 53 52 C1 1C>
```

Deconstructing this example, the following information can be extracted:

1. String length of the name of the protocol used: Decimal number "19" in hex (0x13).
2. Protocol header: The ASCII string "BitTorrent protocol". Note that the string has 19 characters in it, hence the "19" as the string length above.
3. Reserved extension bytes: `00 00 00 00 00 10 00 01`
4. SHA-1 hash of information dictionary:
   `10 1C 9D 63 21 1C 3C 57 0F FB AD D4 9C 56 49 D3 FB 49 72 73`
5. Peer ID:
   `2D 55 54 31 37 37 30 2D F3 9F FD C7 74 B5 6A 4C 53 52 C1 1C`.
   Note that the string `2D 55 54 31 37 37 30 2D` corresponds to the ACSII string `-UT1770-`. This denotes that a uTorrent client running version 1.7.7 created the .torrent file [Bit08].

3.7.2 Other Messages. Once the handshake is complete, the client and remote peer use the peer wire protocol to exchange messages to coordinate the exchange of file piece data. The standard messages supported by the BitTorrent peer wire protocol are listed below [Bit08]:

- `choke`. Sent by a remote peer to a downloader informing it that no requests for data will be honored until the downloader is unchoked.
- `unchoke`. Sent by a remote peer to a downloader informing it that it will honor the downloader's requests for data.
- `interested`. Sent by a client to a remote peer informing it that the remote peer has data the client wishes to download.
- `not interested`. Sent by a client to a remote peer informing it that the remote peer does not have data the client wishes to download.
- `have`. Sent by a downloader when a piece has completed downloading and the hash has been correctly matched to the .torrent file.
- `bitfield`. Sent by a downloader as the first message. This message contains a string of bits representing each piece of the file, with a 1 set for each piece that the downloader possesses, otherwise 0.
- `request`. Sent by a downloader to request a portion of a file piece. This message contains the index of the piece within the file, the offset from the beginning of the piece, and the amount of data requested.
- `piece`. Sent by the peer to the downloader in response to the `request` message. This message contains the index of the piece, the byte offset, and the actual data requested.
- `cancel`. Sent by the downloader to cancel the download of a block of data.
- `keep-alive`. Sent by a peer to keep a connection from being torn down if there is no data being sent in either direction.

3.8 Summary

This chapter provides an overview and discussion of the BitTorrent peer-to-peer protocol. The purpose of the protocol is given, and the services provided by the protocol are discussed. A typical BitTorrent file transfer session is described, and selected terminology and data encoding techniques are discussed. The contents of the .torrent file are analyzed, and the operations of the tracker protocol and peer wire protocol are explained. In particular, the peer wire protocol's handshake packet is identified and analyzed by the forensic tool in order to extract the unique 20-byte hash of the file to be transferred. Finally, real-world examples of BitTorrent messages are examined, and their contents analyzed.

IV. The Session Initiation Protocol

This chapter presents an overview of the Session Initiation Protocol (SIP), which is the other peer-to-peer protocol that is used extensively in this research. To provide a foundation for understanding SIP, Section 4.1 gives an overview of how the protocol functions to make multimedia session connections. Section 4.2 discusses specific terminology necessary for an understanding of the protocol. Finally, Section 4.3 describes how SIP request and response messages are created, and examples of the two request messages used in the research are analyzed.

4.1 Overview of How SIP Works

4.1.1 The Purpose of SIP. In 1999, Henning Schulzrinne of Columbia University submitted the plan for a protocol to establish and control multiparty multimedia sessions, and was approved by the IETF as RFC 2543, the Session Initiation Protocol [Ubi08]. According to the updated version of the protocol, IETF RFC 3261, "SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls" [RFC02]. The goal of SIP is not to exchange data between participants; rather, its purpose is to allow the participants to find one another, and to manage the data connection one established. This allows SIP to used for a large number of data transfer applications, such as interactive gaming, media on demand, and voice or video conferencing [Ubi08].

The Session Initiation Protocol provides for the following basic requirements for setting up, managing, and tearing down communication sessions [RFC02]:

- User Location: Finding the address of the end system to be used for the communication.
- User Availability: Determining if the party being called is willing to join the communication.
- User Capabilities: Determining what type of communications both parties use for communication.

- Session Setup: Establishment of the communication session parameters for both parties.
- Session Management: Governing the transfer of data, terminating communication sessions, and modifying session parameters.

Because SIP is an open source protocol, it is rapidly becoming the de facto standard for multimedia session control. SIP is currently used by the popular VoIP provider Vonage [Cis02], by Microsoft for its MSN Messenger system [Ubi08], and by Yahoo! for its Yahoo! Messenger system [Cor05]. SIP has also been selected by the 3G Community to be its session control protocol for the 3G cellular network [Ubi08], and Google is planning to incorporate SIP into the protocol used by its popular Google Talk service [Goo08].

4.1.2 Making a VoIP Call Using SIP. Figure 4.1 shows how the Session Initiation Protocol is used to make a phone call from one VoIP client to another VoIP client. As shown in the figure, when making a SIP call, the following occurs [RFC02]:

1. The call originator's SIP client sends an INVITE request to the proxy server.
2. The proxy server queries the registrar server for the address of the recipient's SIP client. At the same time, the proxy server sends a message to the call originator indicating that it is attempting to connect the call.
3. The registrar server responds to the proxy server with an IP address where the recipient can be reached.
4. The proxy server forwards the INVITE message to the call recipient.
5. The recipient's SIP client responds that the phone is ringing and is awaiting the recipient to accept the call. The message also contains the IP address and port number of the call recipient for a direct connection at a later time.
6. When the recipient accepts the call, a message is sent from the recipient's SIP client to the proxy server indicating that the call has been answered.

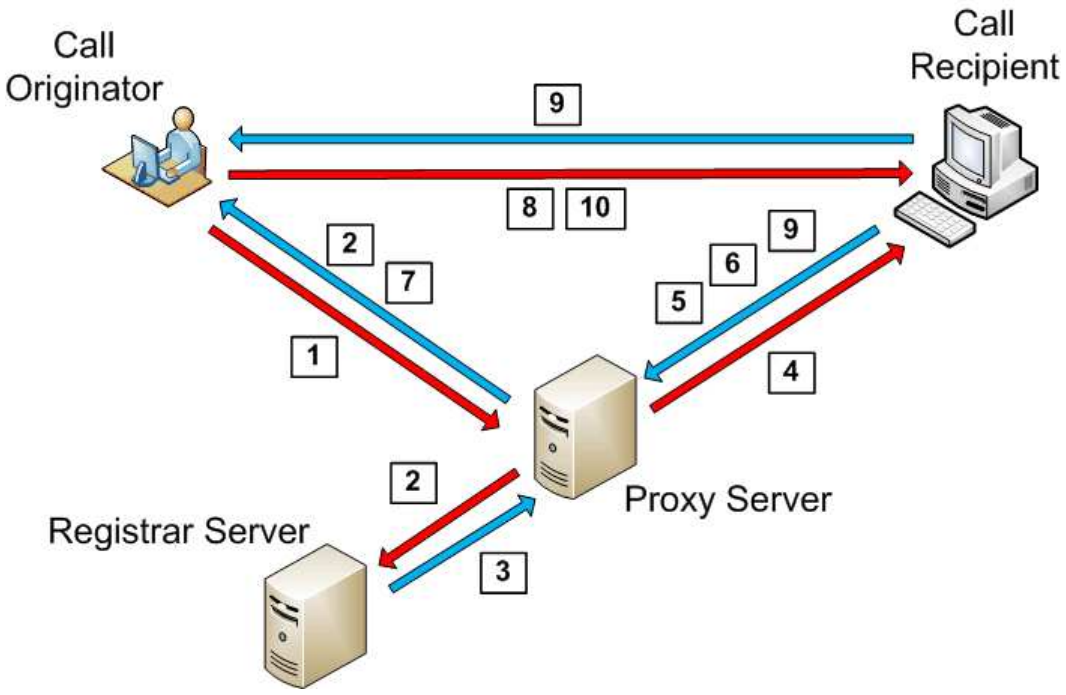

Figure 4.1: How the Session Initiation Protocol Works

7. The proxy server forwards the call answer message to the originator's SIP client.
8. The call originator's SIP client then sends a call acknowledgment directly to the call recipient's SIP client, completing the three-way handshake and beginning the communication session.
9. After the call is completed, one of the parties hangs up, generating a BYE message to the other caller and to the proxy server.
10. The other caller sends an confirmation of the BYE message, ending the SIP connection.

4.2 Terminology

The Session Initiation Protocol uses a specific lexicon to describe different aspects of the protocol's components and functions. Outlined below are some of the most common terms when describing how SIP works, and are used throughout this chapter [RFC02].

4.2.1 Call. A *call* is an informal term that describes some kind of communication between peers that is set up using the SIP protocol. The communication may consist of any type of voice-only or multimedia conversation.

4.2.2 Message. A *message* is a set of data sent between elements of the SIP protocol. *Messages* are usually in the form of either *requests* or *responses*.

4.2.3 Proxy Server. A *proxy server* is a machine that performs the functions of both a server and a client for the purpose of routing requests between clients and servers. When it receives a *request* from a client, the *proxy server* queries the *registrar server* to retrieve the message recipient's addressing information, then forwards the request to the recipient on behalf of the sender.

4.2.4 Registrar Server. A *registrar server* contains a database of the locations of all SIP clients within a network domain. When a call is initiated by a client, the client's *proxy server* will query the *registrar server* and retrieve the recipient's IP address, if it resides in the same domain as the sender.

4.2.5 Requests and Responses. A *request* is a SIP message that is sent from a client to a server in order to cause the server to perform some action. A *response* is a SIP message sent from a server to a client that contains the status of the *request*.

4.2.6 SIP Uniform Resource Identifier. A *SIP Uniform Resource Identifier* (URI) is the address where a SIP client can be found on a network. It uses the same `user@host` format as an email address, and usually contains a user name and a host name. In SIP messages, the *SIP URI* is prefaced by the `sip:` identifier. For example, `sip:alice@afit.edu` and `sip:2001@10.1.1.1` are both valid *SIP URIs*. For the purposes of this research, the terms *SIP URI* and *phone number* are interchangeable.

4.3 SIP Messages

4.3.1 Request Messages. SIP request messages are plain text messages used to communicate information from a client to a server in order to invoke some action by the server. The SIP specification defines six types of request messages, called request methods [RFC02]:

- REGISTER: Registers a client's contact information with a registrar server.
- INVITE: Used to initiate a SIP connection from one client to another.
- ACK: Sent by the call initiator to confirm the reception of a response message.
- CANCEL: Cancels a pending SIP request.
- BYE: Used to terminate a SIP connection session.
- OPTIONS: Used for querying servers about their capabilities.

All SIP request message headers use the same general format. The first line of the header must contain the `Request-Line` field, which contains the request method, the recipient's address, and the protocol version. All other header fields may be placed in any order. Listed below are some of the most common fields found in SIP request messages [RFC02]:

1. `Request-Line`. A string containing a request method, the SIP URI of the message's recipient, and the SIP version number.
2. `Via`. A string containing the SIP version number and transport protocol used (TCP or UDP), the address where the sender expects responses to be sent, and a branch parameter which is a unique alphanumeric string that uniquely identifies the request transaction.
3. `Max-Forwards`. This is the maximum number of hops a request can transit en route to the destination address. The integer is decremented by one every time it transits a router. If the counter reaches 0 before reaching its destination, the

request will be rejected. For most clients and servers, this value is initially set to 70 [RFC02].

4. `Contact`. This field contains the SIP URI that represents a direct route to the sender's SIP client.
5. `To`. This field specifies the SIP URI of the request message's intended recipient. The field may also contain the display name associated with the recipient's SIP URI.
6. `From`. This field specifies the SIP URI of the request message's originator. Like the `To` field, this may contain both the display name and the SIP URI of the sender.
7. `CSeq`. A string that consists of an integer that serves to order transactions and the request method from the `Request-Line`. The start number may be any arbitrary 32-bit unsigned integer less than 2^{31}, and is incremented by one for each additional transaction message sent during the call.
8. `Call-Id`. An alphanumeric string that acts as a unique identifier for the series of messages associated with a single call. This string must be the same for all messages sent during the call. The RFC also recommends that the strings be cryptographically random identifiers to avoid possible session hijacking and unintentional `Call-Id` collisions [RFC02].
9. `Allow`. A string containing the methods supported by the SIP client that is sending the message.
10. `Content-Type`. A string containing the description of the message body.
11. `User-Agent`. This field contains information about the client that sent the message.
12. `Content-Length`. An integer representing the number of bytes contained in the message body.

In this research, the INVITE and BYE requests are captured and examined by the experimental apparatus to determine the originator and recipient of the SIP call. Examples of these two request messages are described in detail below.

4.3.1.1 INVITE Request Example. The following INVITE message is taken from an actual SIP call made from one X-Lite VoIP soft phone [Cor08] to another X-Lite soft phone via an Trixbox registrar/proxy server [Tri08]. This particular message is from the originator at SIP URI 2002@10.1.1.50 and is sent to the SIP URI 2001@10.1.1.50. The packet's header is shown below:

```
INVITE sip:2001@10.1.1.50 SIP/2.0
Via: SIP/2.0/UDP 10.1.1.2:7887;branch=z9hG4bK-d87543-
     bc490705ed79367c-1--d87543-;rport
Max-Forwards: 70
Contact: <sip:2002@10.1.1.2:7887>
To: "2001"<sip:2001@10.1.1.50>
From: "Tester Two"<sip:2002@10.1.1.50>;tag=e07be27e
Call-ID: MTA3ZTZlM2U4ZjRlZDY5ODJmOTMyYTFmMWZiZmFmZTM.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY,
       MESSAGE, SUBSCRIBE, INFO
Content-Type: application/sdp
User-Agent: X-Lite release 1011s stamp 41150
Content-Length: 406
```

Deconstructing this example, the following information can be extracted:

- Destination SIP URI of the message: 2001@10.1.1.50
- SIP version number: 2.0
- IP address and port number to which the sender expects responses to this request: 10.1.1.2:7887

- Number of forwards left in this message: 70
- IP address and port number of a direct route to contact the sender of the message: 10.1.1.2:7887
- Display name and SIP URI of the eventual recipient of the message: "Tester One", 2001@10.1.1.50
- Display name and SIP URI of the originator of the message: "Tester Two", 2002@10.1.1.50
- Type of SIP client used to send the message: X-Lite release 1011s
- Length of the message body: 406 bytes

In this example, the call originator sends the INVITE message to the Trixbox proxy/registrar server at IP address 10.1.1.50, where the server looks up the recipient's address, modifies the INVITE message with the updated information, and forwards the message to the next hop closer to the recipient. Note that the SIP URI in the `From` field contains the IP address of the Trixbox registrar server (10.1.1.50). This allows other SIP clients on the Internet to contact the static address of the registrar server where the sender's client is registered. The `Contact` field, however provides the actual IP address and port number (10.1.1.2:7887) of the SIP client sending the message, which the server will then use to deliver the call recipient's reply to the INVITE message. The `To` field contains the SIP URI of the INVITE message's intended recipient, which in this case is on the same domain as the sender.

4.3.1.2 BYE Request Example. The following BYE message is taken from the same SIP call that generated the INVITE message above (note the identical `Call-ID` fields). This particular message is from the originator at SIP URI 2001@10.1.1.50 and is sent to the SIP URI 2002@10.1.1.50. The packet's header is shown below:

```
BYE sip:2002@10.1.1.2:7887 SIP/2.0
Via: SIP/2.0/UDP 10.1.1.50:5060;
```

```
     branch=z9hG4bK46506adf;rport
From: "2001"<sip:2001@10.1.1.50>;tag=as1b5be240
To: "Tester Two"<sip:2002@10.1.1.50>;tag=e07be27e
Call-ID: MTA3ZTZlM2U4ZjRlZDY5ODJmOTMyYTFmMWZiZmFmZTM.
CSeq: 102 BYE
User-Agent: Asterisk PBX
Max-Forwards: 70
Content-Length: 0
```

In this example, the `Via` field shows that this message is actually from the Trixbox server at IP address 10.1.1.50:5060, and is being forwarded to SIP URI 2002@10.1.1.50 on behalf of SIP URI 2001@10.1.1.50. The fact that the `User-Agent` is an Asterisk PBX confirms this fact.

4.3.2 Response Messages. Response messages differ from request messages in that they have a status line as the first line of the message. The status line consists of the SIP version followed by a Status-Code and its Reason-Phrase. Table 4.1 describes the main types of Status-Codes. Note that the Status-Codes used by SIP are almost identical to those used by HTTP, with the exception of the 6xx codes [RFC99].

Table 4.1: Status Codes for SIP Response Messages [RFC02]

Status-Code	Response Type	Meaning
1xx	Provisional	Request received, continuing to process the request
2xx	Success	The action was successfully received and accepted
3xx	Redirection	Further action needs to be taken to complete the request
4xx	Client Error	Request contains bad syntax or can't be fulfilled by the server
5xx	Server Error	Server failed to fulfill an apparently valid request
6xx	Global Failure	Request cannot be fulfilled at any server

Other than the first line of the message header, response message headers contain the same fields as request messages. When a client or server generates a response to a request, the response code is placed in the first line of the header, and the `Via, To, From, Call-ID,` and `CSeq` fields are copied from the request header to the

response header. The `Contact` field is then added which contains the SIP URI that represents a direct route to the client or server that generated the response.

4.4 Summary

This chapter provides an overview and discussion of the Session Initiation Protocol. The purpose of the protocol is given, and the services provided by the protocol are discussed. A typical SIP session is described, and selected terminology is defined. SIP request and response messages are analyzed, and real world examples of the INVITE and BYE messages are deconstructed. In particular, the SIP INVITE and BYE packets are identified and analyzed by the forensic tool in order to extract the caller and receiver SIP URIs, and are also used to determine the beginning and end of the SIP session.

V. Methodology

This chapter presents the methodology used to evaluate the performance of the TRAPP system using two different metrics: the time required to process a packet, and the probability of packet intercept under high network utilization. First, the problem definition, goals and hypotheses, and approach are discussed in Section 5.1. Section 5.2 defines the system boundaries. The system and its services are described in Section 5.3, followed by a detailed description of workload in Section 5.4, performance metrics in Section 5.5, parameters in Section 5.6, and factors in Section 5.7. Then, the evaluation technique is discussed in Section 5.8 followed by a description of the experimental design in Section 5.9. Finally, the techniques used to analyze and interpret the data are covered in Section 5.10.

5.1 Problem Definition

5.1.1 Goals and Hypotheses. The objective of this thesis is to develop a system to identify and track specific digital information being transmitted on a network using peer-to-peer protocols. The proposed system will detect peer-to-peer transmissions on a target network, classify them by specific peer-to-peer protocol, compare the digital file being transmitted against a list of interest, and identify the sender and recipient by Internet Protocol (IP) address.

The goals of this research are to:

- Construct an FPGA-based system that analyzes traffic on a network, detects a selected peer-to-peer protocol, compares the digital information being shared against a list of interest, and in the case of a match, records selected control packets ("packets of interest") from the peer-to-peer session in a log file.
- Optimize the system such that it is able to detect and record all packets of interest on the network, even under a heavy (approximately 90 percent utilization) non-peer-to-peer traffic load.

- Modify the system to detect and record control packets of interest belonging to a second peer-to-peer protocol with no negative impact on overall performance.

It is hypothesized that a system equipped for a single peer-to-peer protocol can be constructed and optimized such that a packet of interest is detected and recorded on a network with 90% utilization with at least 95% probability. It is also hypothesized that the system can be expanded to include a second peer-to-peer protocol while maintaining at least a 95% probability of intercept for a packet of interest from either peer-to-peer protocol.

5.1.2 Approach. A hardware-based forensic tool is designed using the Virtex II Pro FPGA development board [Xil08b] and the BitTorrent peer-to-peer protocol. Implementing the system on an FPGA allows the software application to directly access the Ethernet controller buffers, bypassing the rest of the network stack and increasing the system's simplicity and speed. Once the system is optimized and tested using this protocol, the system is expanded to also process the Session Initiation Protocol, and tested again. Details on the hardware construction of the system are in Appendix A.

Figure 5.1 shows the overall functionality of the design. When the system processes a packet, the following occurs:

1. The tool fingerprints the frame received from the network by extracting the first 32 bits of the frame's payload.
2. The 32-bit fingerprint is then compared to the first 32 bits of a BitTorrent Handshake message, which is `0x13426974` [Bit08], a SIP INVITE message, which is the ASCII string "`INVI`", or a SIP BYE message, which is the ASCII string "`BYE `" [RFC02].
3. If the first word of the frame's payload is not a match to any of these strings, the frame is discarded.

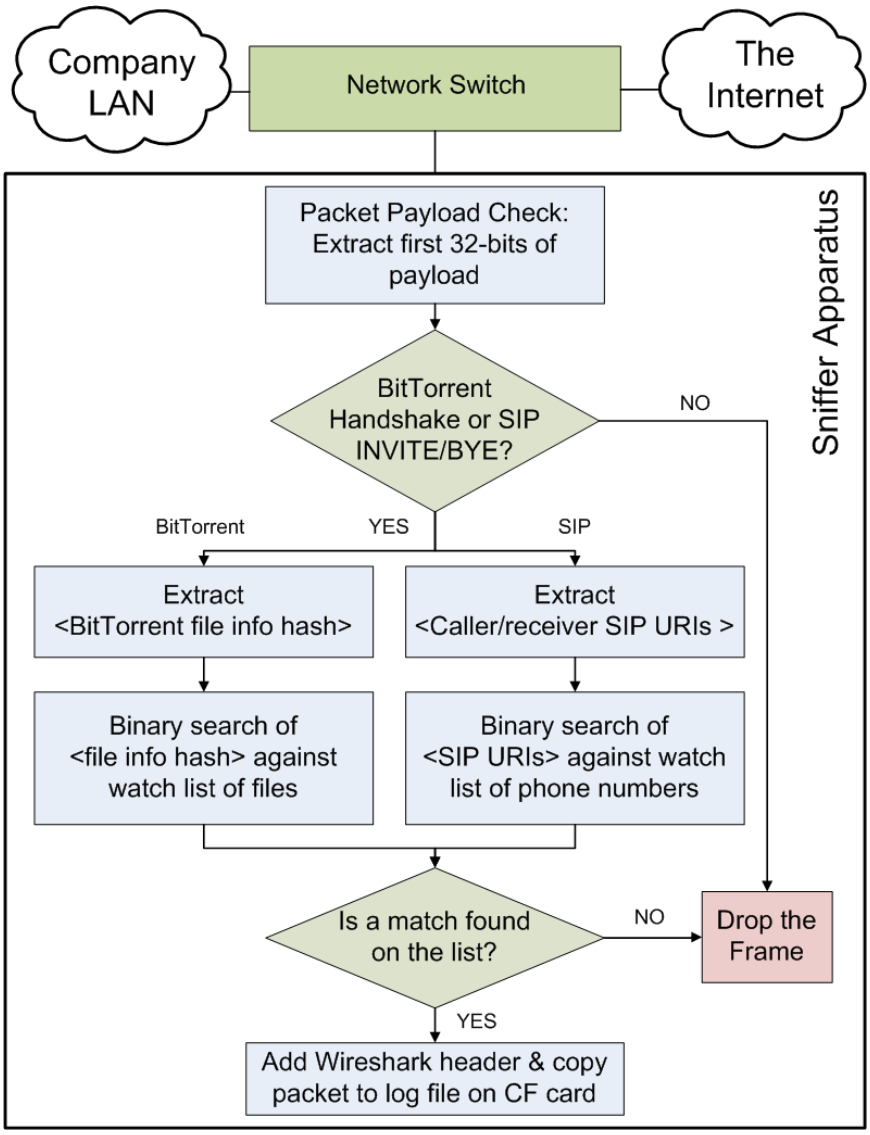

Figure 5.1: Packet Data Flow through the TRAPP System

4. If the word matches that of a BitTorrent Handshake message, the first 32 bits of the Handshake's file info hash is extracted from the frame, and compared against a list of hashes belonging to files of interest using a binary search.

5. If the word matches that of a SIP message, the first 12 characters of the TO and FROM SIP URIs are extracted from the frame, and each is compared against a separate list of SIP URIs of interest using a binary search.

6. If the file info hash/SIP URI is not on the list, the frame is dropped.

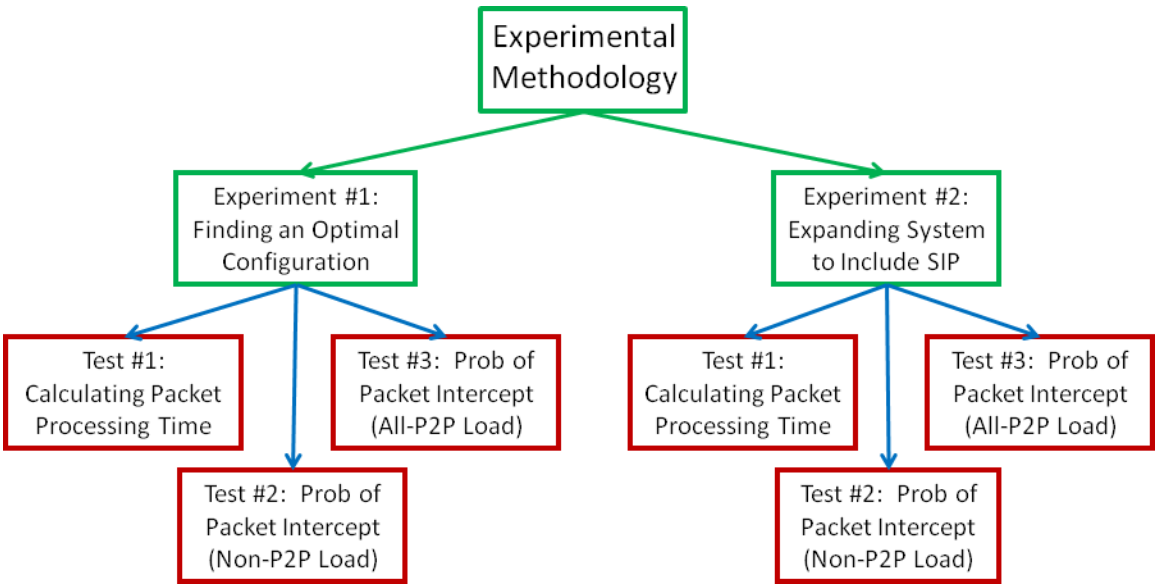

Figure 5.2: Experiments and Tests Used to Achieve the Research Goals

7. If the file info hash/SIP URI is on the list, the frame is saved in a Wireshark-readable log file and placed on a Compact Flash card. The frames recorded in the log file are later analyzed to extract IP address information, which can then be used to perform tracking and forensic analysis.

As shown in Figure 5.2, this research is divided into two experiments, finding a software configuration for the system that processes BitTorrent packets of interest as quickly as possible, and expanding the system to incorporate the SIP protocol without sacrificing overall performance. Each of the two experiments is comprised of three tests: calculating packet processing time, calculating probability of packet intercept under a non-peer-to-peer workload, and calculating probability of packet intercept under an all-peer-to-peer workload. Overviews of the two experiments are outlined below.

5.1.2.1 Experiment 1: Finding an Optimal Software Configuration. The first experiment seeks to determine the optimal hardware/software configuration of the system that processes BitTorrent packets of interest as quickly and as accurately as possible. This experiment is split into three parts. First, each hardware/software

configuration is tested against several types of packet sizes and formats, and the amount of processor time needed to process each packet is examined. Second, a series of BitTorrent packets of interest are sent to each configuration in a high non-peer-to-peer network utilization environment, and the overall probability of intercept of a packet of interest is calculated for each configuration. Finally, a series of BitTorrent packets is sent to the system at near-full network utilization in order to determine the probability of intercepting multiple packets of interest in a row.

5.1.2.2 Experiment 2: Expanding the Forensic Tool to Incorporate VoIP Functionality. The second experiment seeks to determine if adding functionality to process SIP packets in addition to BitTorrent packets degrades overall system performance. For this experiment, the optimal configuration found in the first experiment is modified to also include detection and processing of SIP packets of interest. As with the first experiment, this experiment consists of three parts. First, the modified configuration is tested against several types of BitTorrent and SIP packets, and the amount of time needed to process each packet is examined. Second, a series of BitTorrent and SIP packets of interest are sent to the modified configuration in a high network utilization environment, and the probability of intercept of each type of packet of interest is calculated. The results of this experiment are then compared against the results of the first experiment to determine if the system's overall performance in processing BitTorrent packets of interest is negatively impacted. Finally, a series of peer-to-peer packets is sent to the system at near-full network utilization in order to determine a measure of the probability of intercepting multiple packets of interest in a row for each peer-to-peer protocol.

5.2 System Boundaries

The System Under Test (SUT) is the TRAPP Forensic Tool System. A block diagram of the SUT is shown in Figure 5.3. It consists of the following components: the TRAPP software, the Power PC processor, the system clock, the Ethernet con-

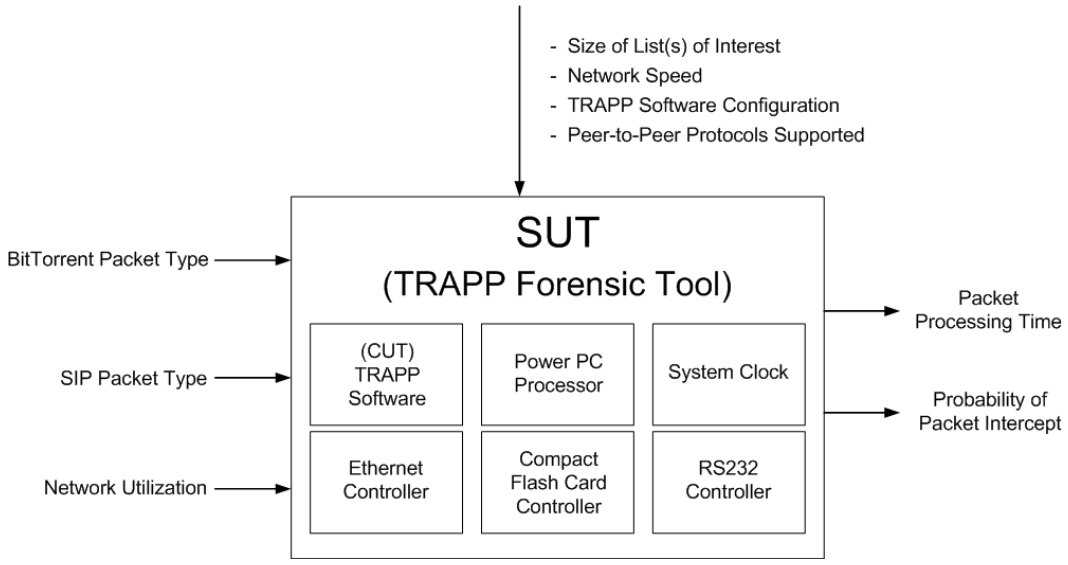

Figure 5.3: The TRAPP Forensic Tool System

troller, the Compact Flash card controller, and the RS232 controller. The Component Under Test (CUT) is the TRAPP software. Specifically, various modifications to the software execution flow will be compared to the baseline software architecture.

Workload parameters include the type of BitTorrent Packet used, the type of SIP packet used, and the total network utilization as a percentage of network capacity. The system parameters consist of the size of the list of interest used by the system, network speed, the software configuration of the TRAPP system, and the number and types of peer-to-peer protocols supported by the system. These parameters are discussed in more detail in Section 5.6. The metrics of the system consist of the time required to process a packet and the probability of successful intercept of a packet.

5.3 System Services

This system provides a detection and tracking service for certain peer-to-peer protocols on a local area network gateway to the Internet. The service is successful when a BitTorrent Handshake packet, SIP INVITE packet, or SIP BYE packet entering the system is detected, its file info hash or SIP URIs extracted and compared against a list of interest, and in the case of a match, its contents copied to

a Wireshark-compatible log file. The service is a failure when: a packet of interest is not detected, a packet of interest is detected but the file info hash or SIP URIs are not extracted correctly, the file info hash or SIP URI is incorrectly determined to be not on the list, or the packet fails to be copied correctly to the log file. For this research, the problem of encountering false positives (for example, the file info hash or SIP URI is determined to be a match to an item on the list, when in fact it is not) is not considered, as it is assumed that all log files will be reviewed by a human administrator, who will be able to correctly determine that the packet is not of interest.

5.4 Workload

The workload of the SUT consists of two parts, the peer-to-peer packets and the non-peer-to-peer network traffic load. For the two experimental tests, either one or both of these workload components are used. Outlined below are the specific peer-to-peer packets used in the tests and a description of how the non-peer-to-peer traffic is generated.

5.4.1 BitTorrent and SIP Packets Employed. The peer-to-peer packets employed for the system's workload consist of BitTorrent Handshake packets and SIP INVITE and BYE packets that are loaded with specific file info hashes and SIP URIs. Listed below are the variations of these packets that comprise the peer-to-peer portion of the workload.

5.4.1.1 BitTorrent Handshake Packet (Hash on the List). To determine how the system processes packets that belong to the BitTorrent protocol, and whose file info hashes are contained in the list of interest, a packet capture file containing a real-world BitTorrent file transfer is analyzed, and the payload of the Handshake packet used to set up the peer-to-peer connection is extracted. The file info hash contained in the Handshake packet is then added to the list of interest. The contents of the sample BitTorrent Handshake packet are shown below.

```
<13 42 69 74 54 6f 72 72 65 6e 74 20 70 72 6f 74 6f 63 6f 6c
 00 00 00 00 00 01 00 01
 3f fe ce 46 da 61 36 15 2f 25 59 69 07 4a d9 5b 07 fa 08 75
 2d 55 54 31 37 37 30 2d f3 9f b2 53 84 dc 26 f6 32 05 e7 be>
```

Details on how to interpret the contents of the BitTorrent Handshake packets presented here are in Section 3.7.

5.4.1.2 BitTorrent Handshake Packet (Hash Not on the List). To determine how the system processes packets that belong to the BitTorrent protocol, but whose file info hashes are not on the list of interest, the BitTorrent Handshake packet from the previous section is modified such that the new file info hash does not match any entry on the system's list of interest. As shown in the packet payload below, the third byte of the file info hash (contained in the third line of the contents) is changed from `0xCE` to `0xAA`.

```
<13 42 69 74 54 6f 72 72 65 6e 74 20 70 72 6f 74 6f 63 6f 6c
 00 00 00 00 00 01 00 01
 3f fe aa 46 da 61 36 15 2f 25 59 69 07 4a d9 5b 07 fa 08 75
 2d 55 54 31 37 37 30 2d f3 9f b2 53 84 dc 26 f6 32 05 e7 be>
```

5.4.1.3 Non-Peer-to-Peer Packet. To determine how the system processes packets that do not belong to one of the two peer-to-peer protocols of interest, a packet is crafted that does not match the parameters of either protocol. As shown in the packet payload below, this is accomplished by taking a BitTorrent Handshake packet and changing the first byte of the payload from `0x13` to `0xAA`. This change guarantees that the packet does not match the format of either a BitTorrent packet or a SIP packet (the packet does not match any known SIP packet since all SIP packet payloads are written in HTML, which is an ASCII-based language, and the byte 0xAA is not a valid ASCII character).

```
<aa 42 69 74 54 6f 72 72 65 6e 74 20 70 72 6f 74 6f 63 6f 6c
```

```
00 00 00 00 00 01 00 01
3f fe ce 46 da 61 36 15 2f 25 59 69 07 4a d9 5b 07 fa 08 75
2d 55 54 31 37 37 30 2d f3 9f b2 53 84 dc 26 f6 32 05 e7 be>
```

5.4.1.4 SIP INVITE Packet (SIP URI on the List). To determine how the system processes packets that belong to the SIP protocol, and whose TO and FROM SIP URIs are contained in the list of interest, a packet capture file containing a real-world SIP phone call setup and teardown is analyzed, and the payload of the INVITE packet used to set up the phone call is extracted. The SIP URIs contained in the packet are then added to the list of interest. The contents of the sample SIP INVITE message header are shown below. Lines 6 and 7 of the payload contain the sender and receiver SIP URIs (`2001@10.1.1.50` and `2002@10.1.1.50`) that are compared against the list of interest.

```
INVITE sip:2001@10.1.1.50 SIP/2.0
Via: SIP/2.0/UDP 10.1.1.2:8228;
branch=z9hG4bK-d87543-ab64a6626128d369-1--d87543-;rport
Max-Forwards: 70
Contact: <sip:2002@10.1.1.2:8228>
To: "2001"<sip:2001@10.1.1.50>
From: "Tester Two"<sip:2002@10.1.1.50>;tag=48662f73
Call-ID: NGM2ZjNmNDkyZGM1NmFmNmUxOGYwZWM5YjU5MTFhMGU.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO
Content-Type: application/sdp
User-Agent: X-Lite release 1011s stamp 41150
Content-Length: 408
```

Further details on the format of the SIP INVITE packet are found in Section 4.3.

5.4.1.5 SIP INVITE Packet (SIP URI Not on the List). To determine how the system processes packets that belong to the SIP protocol, but whose TO and FROM SIP URIs are not on the list of interest, a SIP INVITE packet whose SIP URIs are on the list of interest is modified such that the new SIP URIs not match any entry on the experiment's list of interest. As shown in the SIP message header below, the first four bytes of the TO and FROM SIP URIs (contained in lines 6 and 7 of the contents) are changed from 2001 and 2002 to 9999, which is not on the list of interest.

```
INVITE sip:2001@10.1.1.50 SIP/2.0
Via: SIP/2.0/UDP 10.1.1.2:8228;
branch=z9hG4bK-d87543-ab64a6626128d369-1--d87543-;rport
Max-Forwards: 70
Contact: <sip:9999@10.1.1.2:8228>
To: "2001"<sip:9999@10.1.1.50>
From: "Tester Two"<sip:9999@10.1.1.50>;tag=48662f73
Call-ID: NGM2ZjNmNDkyZGM1NmFmNmUxOGYwZWM5YjU5MTFhMGU.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO
Content-Type: application/sdp
User-Agent: X-Lite release 1011s stamp 41150
Content-Length: 408
```

5.4.1.6 SIP BYE Packet (SIP URI on the List). To determine how the system processes packets that belong to the SIP protocol, and whose TO and FROM SIP URIs are contained in the list of interest, the same packet capture file used above to extract the INVITE packet is again analyzed, and the payload of the BYE packet used to tear down the phone call is extracted. The contents of the sample SIP BYE message header are shown below.

```
BYE sip:2002@10.1.1.50 SIP/2.0
Via: SIP/2.0/UDP 10.1.1.1:7946;
branch=z9hG4bK-d87543-181b4b34431d0f61-1--d87543-;rport
Max-Forwards: 70
Contact: <sip:2001@10.1.1.1:7946;rinstance=c1540086ad2a9650>
To: "SANS Student 2"<sip:2002@10.1.1.50>;tag=as1b3919be
From: <sip:2001@10.1.1.1:7946;rinstance=c1540086ad2a9650>;tag=d33db83e
Call-ID: 73a1d1f673e3e4b42cfb4cca38f30aa1@10.1.1.50
CSeq: 2 BYE
User-Agent: X-Lite release 1011s stamp 41150
Reason: SIP;description="User Hung Up"
Content-Length: 0
```

Further details on the format of the SIP BYE packet are found in Section 4.3.

5.4.2 The Non-Peer-to-Peer Traffic Load. For the probability of intercept tests in each of the two experiments, a non-peer-to-peer traffic load is used to saturate the test network to approximately 90 Mbps on the 100 Mbps network. To create the load, a 1,150,217,528-byte video file, named DFEE-660.avi, is transferred from one node on the network to another node on the network using the Windows NETBIOS file transfer protocol. While this transfer is in progress, another node on the network can transmit peer-to-peer packets of interest into a high-traffic environment.

5.5 Performance Metrics

In order for the system to be effective, it must have a high probability of successfully intercepting, processing, and recording those packets on the network that belong to a peer-to-peer protocol supported by the system, and whose identifiers are on the lists of interest. By extension, in order for the system to successfully intercept these packets of interest, it must have the capability to analyze all traffic on a network, which necessitates the requirement of processing each packet as quickly as possible for a given set of parameters. Thus, the following performance metrics are defined:

- *Packet Processing Time*: The number of CPU cycles, as measured by the Power PC processor's System Timer, that are required to accomplish the following: determine if a packet has been received by Ethernet controller, inspect the packet for peer-to-peer protocols, match the packet's identifier against the appropriate list, record the packet if necessary, and make the Ethernet controller available to receive another packet entering the system.
- *Probability of Packet Intercept*: The probability of a packet, whose format matches a peer-to-peer protocol supported by the system, and whose identifier matches an entry on a list of interest, being successfully recorded in the system intercept log.

5.6 Parameters

The parameters of the system are the properties which, when changed, impacts the performance of the system. These include both system parameters, which characterize the system, and workload parameters, which characterize the workload. The system and workload parameters for the SUT are described below.

5.6.1 System Parameters.

- Size of List of Interest: This is the size of the list of interest, expressed by the number of entries in the list. For the BitTorrent protocol, an entry is a 160-bit file info hash, described in Section 3.5.3. For the SIP protocol, an entry is the first 12 digits of a SIP-URI, described in Section 4.2. Because the system uses a binary search algorithm to perform the hash/SIP URI matching process, each doubling of the list size will add a maximum of one comparison to the total algorithm execution time. For this research, a sample list size of 1000 entries is used for both the file info hash list and the SIP URI list.
- Network Speed: This is the maximum speed of network data entering the system through the Ethernet connection. The Ethernet controller on the Xilinx II Pro

board used in this research is capabable of connecting to either a 10 Mbps or a 100 Mbps network. For this research, the 100 Mbps connection option is used.

- TRAPP Software Configuration: The software code used to execute TRAPP functions using the Power PC processor on the FPGA. As procedures and features contained in the software are added, removed, or modified, the overall functions and performance of the system are affected. A full description of each configuration used in the TRAPP system is contained in Section 5.7.
- Peer-to-Peer Protocols Supported: This is the set of peer-to-peer protocols that the system is capable of detecting and analyzing. For the first experiment, the BitTorrent protocol is the only member of this set. In the second experiment, the Session Initiation Protocol is added to the set.

5.6.2 Workload Parameters.

- BitTorrent Packet Type: In this study, three different types of BitTorrent packets are used: a BitTorrent Handshake packet whose file info hash matches an entry on the list of interest, a BitTorrent Handshake packet whose file info hash does not match an entry on the list of interest, and a packet that is not a properly formatted BitTorrent Handshake packet. For a complete description of the packets, see Section 5.4.1.
- SIP Packet Type: Three different types of SIP packets are used in this study: a SIP INVITE packet whose TO and FROM SIP URIs match an entry on the list of interest, a SIP INVITE packet whose TO and FROM SIP URIs do not match an entry on the list of interest, and a SIP BYE packet whose TO and FROM SIP URIs match an entry on the list of interest. A complete description of the packets is in Section 5.4.1.
- Network Utilization: This is the total amount of traffic entering the system. For the first test, the network utilization is limited to single packets injected into the system to measure the time required to fully process the packet. For

the second test, a load of between 89.6 Mbps and 89.7 Mbps is injected into the system, which equates to approximately a 90% utilization of the 100 Mbps Ethernet connection. For the third test, a continuous stream of identical peer-to-peer packets of interest is injected into the system, with the network utilization varying with the type of peer-to-peer packet.

5.7 Factors

This section outlines the factors selected from the system and workload parameters. These factors are varied in Experiment 1 to determine the software configuration that returns the best performance in processing various types of BitTorrent packets. They are varied in Experiment 2 to measure the impact on performance of adding a second peer-to-peer protocol to the optimal configuration found in Experiment 1. Table 5.1 and Table 5.2 summarize the factors chosen and their levels for the two experiments.

Table 5.1: Factor Levels for Experiment 1

Factor	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Configuration	Control	User Alerts	Dual Buffer	Packet Write	Cache	Combined
Packet Type	Non-P2P	BT On List	BT Off List			

Table 5.2: Factor Levels for Experiment 2

Factor	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Configuration	Optimized (BT + SIP)					
Packet Type	Non-P2P	BT On List	BT Off List	SIP INVITE On List	SIP INVITE Off List	SIP BYE On List

5.7.1 Configuration. The software configuration is the factor designated as the Component Under Test. It controls all aspects of the system, including what information is provided to the user, how network data is captured and analyzed, and how packets of interest are stored. The six levels chosen for this factor are detailed below.

5.7.1.1 Control Configuration. In order to design and build a prototype as quickly as possible, the system is initially implemented as an embedded software application using the Power PC core on the Virtex II Pro FPGA development board. Xilinx-provided drivers and built-in functions are used where possible, with custom software built to accomplish the following functions: read the data file containing the file info hashes of the list of interest, perform packet payload inspections, copy BitTorrent Handshake frames to on-chip RAM, perform the hash matching, and write the frame data to the log file on the Compact Flash card.

Listed below are the salient features of the Control configuration:

- All modules are executed in software. The only hardware modification made is to enable the Ethernet controller to operate in promiscuous mode (see Section A.3 for details).
- To simplify the software code as much as possible, the Ethernet controller is limited to one receive buffer, caching is not used, and no user alerts are generated for the user.
- Packets of interest are copied three times. The first copy is from the Ethernet controller buffer to block RAM upon detection of the 32-bit BitTorrent signature in the packet's payload. If the file info hash is found on the list, the frame is copied from RAM to a character array, and then from the array to the log file on the Compact Flash card.
- Frames are copied to the Compact Flash card as they are processed. The system waits until the current frame has been completely processed and sent to the Compact Flash card before beginning to process another frame.

5.7.1.2 User Alerts Configuration. This modification adds to the system user notifications, via the serial port and HyperTerminal, of any peer-to-peer control packets that are found by the system. The messages consist of the type of peer-to-peer packet found, whether the file info hash matches an entry on the list of

interest, and the file info hash's position on the list of interest. Because the serial port runs at a much lower speed than the CPU and the processing bus, it is hypothesized that sending any data over the RS232 connection causes a dramatic slowdown in overall processing time.

5.7.1.3 Packet Write Configuration. In this modification, all captured packets of interest are stored within a RAM block instead of writing them individually to the Compact Flash card. When the system is shut down, all data is then transferred from the block RAM to the Compact Flash card. By storing the data within RAM, the only write functions to the Compact Flash card are performed before packet sniffing begins and after packet sniffing terminates. It is hypothesized that writing to the Compact Flash card is a high-latency process, and that its removal will result in a significant processing time savings.

5.7.1.4 Dual Buffer Configuration. This modification adds a second receive buffer to the Ethernet controller [Pro06]. This allows one frame to be read and processed while another frame is received. The goals for this optimization are to give the comparison and copying routines additional time to execute, and limit the number of frames dropped due to a full receive buffer.

5.7.1.5 Cache Configuration. This modification enables the instruction and data caches for the Power PC processor. By allowing the FPGA to cache processor instructions, heap data, and stack data instead of performing multiple reads and writes to block RAM, a significant amount of processing time should be saved.

5.7.1.6 Combined Configuration. This is the combined case of the CUT incorporating the Packet Write, Dual Buffer, and Cache optimizations into a single system. The goal for the integration is to take advantage of each optimization individually and to possibly gain synergistic time savings from the combination of all four optimizations.

5.7.2 Packet Type. This is the format, protocol, and size of the packets entering the system. A sufficient cross section of peer-to-peer packet sizes and types are crucial to determining how the system will process various packets of interest and packets not of interest. The six levels chosen for this factor are listed below. Detailed descriptions of each packet format are found in Section 5.4.1.

- Non-Peer-to-Peer Packet
- BitTorrent Handshake Packet On the List of Interest
- BitTorrent Handshake Packet Not On the List of Interest
- SIP INVITE Packet On the List of Interest
- SIP INVITE Packet Not On the List of Interest
- SIP BYE Packet On the List of Interest

Note that since a SIP BYE Packet Not on the List of Interest and a SIP INVITE Packet Not on the List of Interest will be treated the same by the system, this type of SIP BYE packet is not included as a factor level. Except for the first line of the message header, each packet has identical header contents, and since the SIP URI of neither packet is on the list, neither frame will be copied by the system to the log file. Thus, the two packet types will be processed identically by the system.

5.8 Evaluation Technique and Environment

5.8.1 Experimental Environment. To conduct the two experiments outlined in Section 5.1.2, the experimental setup shown in Figure 5.4 is created. The experimental environment consists of the following components:

- One Cisco Catalyst 2900XL 100 Mbps switch, configured with 22 standard ports and 2 spanning ports.
- Two Dell Inspiron Windows XP Service Pack 2 laptops loaded with uTorrent 1.7.7 [uTo08], a popular BitTorrent client, and X-Lite 3.0 [Cor08], a popular VoIP phone client, and connected to the switch.

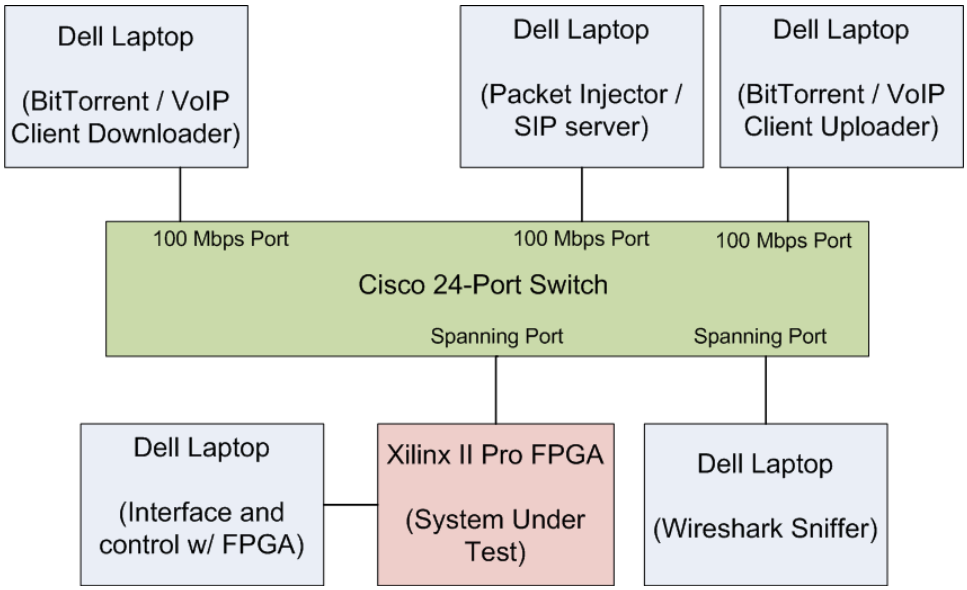

Figure 5.4: Block Diagram of the Experimental Setup

- One Dell Inspiron laptop that is dual-equipped with the BackTrack 2.0 Linux environment [RE08] and Windows XP Service Pack 2, and is connected to the switch. The BackTrack environment contains the Hping 3.0.0 [Hpi08] utility, which is used to inject the crafted BitTorrent and SIP packets. The Windows environment contains the VMWare 2.0.5 [VMW08] utility to run a TrixBox 2.2 [Tri08] SIP proxy and registrar server for use with the X-Lite clients.
- One Virtex II Pro FPGA system (the SUT), which is connected to a spanning port on the switch.
- One Dell Inspiron Windows XP Service Pack 2 laptop loaded with Wireshark 1.0.1 [Wir08], which is connected to a second spanning port as a control packet sniffer.
- One Dell Windows XP Service Pack 2 laptop, which is connected to the SUT and is used to configure and load the Virtex II Pro via USB port. The laptop is also equipped with TTermPro [Pro08a], a HyperTerminal application used to receive alerts from the SUT via RS232 serial port.

  The actual setup used in the experiments is shown in Figure 5.5.

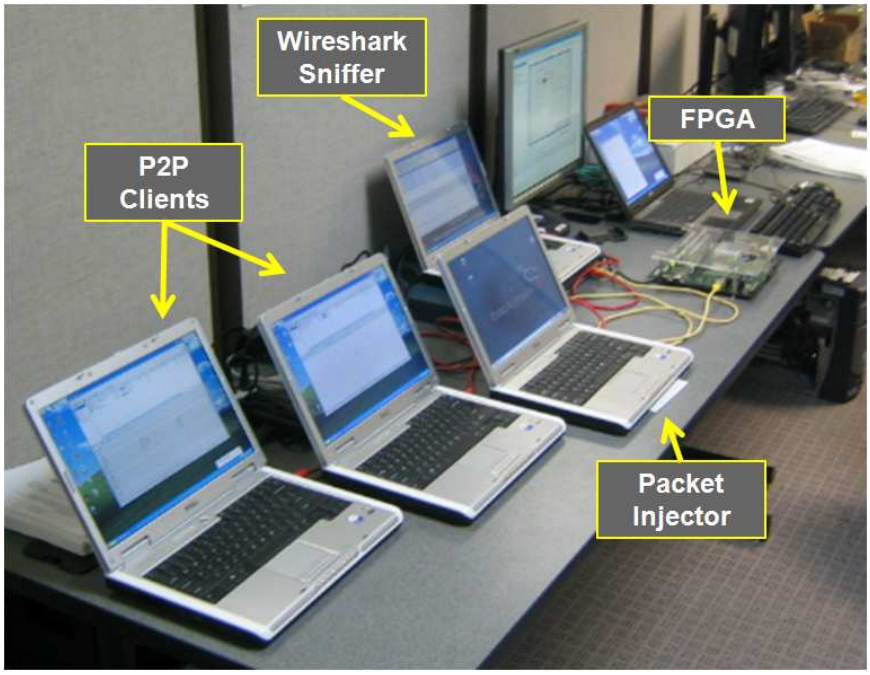

Figure 5.5: Experimental Setup for the Three Performance Tests

5.8.2 Evaluation Techniques. Outlined below are the tests used for collecting data on the *packet processing time* and *probability of packet intercept* metrics for each experiment.

5.8.2.1 Calculating Packet Processing Time. The first test consists of a series of packets sent from the Linux laptop to one of the Windows laptops via the Cisco switch. For each run, a series of 50 crafted packets are sent, and the CPU cycles needed by the system to process each packet is recorded. The crafted packets are described in detail in Section 5.4.1.

A total of 50 identical packets are sent one second apart through the network. Based on the testing of several different sample sizes, 50 packets is the minimum sample size that results in sufficiently small confidence intervals to perform a meaningful comparison between system configurations. To ensure the independence of each trial, one second is chosen as the interarrival interval. To determine the number of cycles required to process each packet received by the system, a Power PC System Timer time stamp is taken prior to the beginning of the processing, and another time stamp

is taken immediately after the processing routine ends. To compute the number of cycles required to process the packet, the two values are simply subtracted from each other. Since the Power PC processor in the SUT is configured to run at 300 MHz, to convert the processing time from clock cycles to standard time units, the formula ($time = cycles/300$) is used, where *cycles* is the number of cycles as determined by the System Timer and *time* is the time to process the packet in microseconds.

5.8.2.2 Calculating Probability of Intercept Under a Non-Peer-to-Peer Load. The second test consists of a series of packets sent from the Linux laptop to one of the Windows laptops via the Cisco switch. For this test, however, an additional non-peer-to-peer traffic load, as described in Section 5.6.2, is generated on the network. For this test, the number of crafted packets successfully intercepted and processed by the system is recorded. The crafted packets used by the Linux laptop are described in detail in Section 5.4.1.

For each test, a series of three hundred crafted packets are injected into the network 500 milliseconds apart. By injecting the packets 500 milliseconds apart, the results of each trial (either the packet was captured or not captured) are assured to be independent of each other. Based on the testing of several different sample sizes, 300 packets is a good sample size to produce a binomial distribution that results in sufficiently small confidence intervals to perform a meaningful comparison between system configurations. Again, to ensure the independence of each trial, i.e., to ensure that the system is not processing one crafted packet when another one arrives at the system, the packets are sent 500 milliseconds apart.

When performing this test, another important parameter is the network utilization or network load, defined in Section 5.4. To determine the minimum overall network load, the Wireshark utility on the laptop that is connected to the second spanning port is used to analyze all traffic sent during the test. At the conclusion of the test, a capture summary is extracted by selecting the `Statistics -> Summary` menu option, and recording the `Average MBit/sec` field from the `Captured` column.

Because Wireshark itself may not capture all packets on the network, this value is assumed to be the *minimum* network traffic load.

5.8.2.3 Calculating Probability of Intercept Under an All-Peer-to-Peer Load. The third test also consists of a series of packets sent from the Linux laptop to one of the Windows laptops via the Cisco switch. As in the first test, a series of crafted packets, described in detail in Section 5.4.1, are sent across the network. However, for this test, the packets are sent as quickly as possible from the Linux laptop using the Hping `--flood` switch.

In order to determine how many packets were sent by the Linux laptop, a particular feature of the Hping program is exploited. When the Hping program sends a series of packets, each packet contains a different source port, and the source port number is incremented by one each time a packet is transmitted. Thus, for a given series of packets, the total number of packets sent in the series can be calculated by subtracting the first packet's source port number from the last packet's source port number.

In this test several thousand packets are sent over the network as quickly as possible using Hping, and the flood is then terminated manually. To determine the *probability of packet intercept*, the following procedure is used:

1. Inspect the capture log file and record the number of frames successfully captured by the system.
2. Record the source port number of the first packet in the log file, and the source port number of the last packet in the log file.
3. Apply the formula

$$P(packet\ intercept) = \frac{number\ of\ packets\ in\ log}{Port(last\ packet) - Port(first\ packet)} \tag{5.1}$$

to compute the *probability of packet intercept*.

When performing this test, another important parameter is the network utilization or network load, defined in Section 5.6.2. To determine the minimum overall network load, the Wireshark utility on the laptop that is connected to the second spanning port is used to analyze all traffic sent during the test. At the conclusion of the test, a capture summary is extracted by selecting the `Statistics -> Summary` menu option, and recording the `Average MBit/sec` field from the `Captured` column. Because Wireshark itself may not capture all packets on the network, this value is assumed to be the *minimum* network traffic load.

5.9 Experimental Design

The overall experimental design for this study consists of two experiments, each with a partial-factorial design with factor levels selected from Tables 5.1 and 5.2. The overall design consists of a total of 3,900 trials, where each trial consists of a crafted test packet sent through the network to the SUT. Details of the trial distribution between the tests of each experiment are outlined below.

5.9.1 Experiment 1: Finding an Optimal Configuration. The first experiment, Finding an Optimal Software Configuration, consists of 900 packets (6 configurations * 3 workloads * 50 packets) sent to the experimental setup to compute the *packet processing times* and 1,800 packets (6 configurations * 1 workload * 300 packets) sent to the setup to compute the *probability of packet intercept* for a non-peer-to-peer workload. To compute the *probability of packet intercept* for an all-BitTorrent workload, several hundred packets are sent for each of the six configurations and the "BT On List" packet type.

The configurations and workloads for this experiment are given in Table 5.1. For the *packet processing time* test, all configuration and workload factor levels are used. For the *probability of packet intercept* (non-peer-to-peer workload) test, all six configurations and the "BT On List" packet type are used. For the *probability of*

packet intercept (all-BitTorrent workload) test, all six configurations and the "BT On List" packet type are used.

5.9.2 Experiment 2: Expanding the System. The second experiment, Expanding the Tool to Incorporate VoIP, consists of 300 packets (1 configuration * 6 workloads * 50 packets) sent to the experimental setup to compute the *packet processing times* and 900 packets (1 configuration * 3 workloads * 300 packets) sent to the setup to compute the *probability of packet intercept*. To compute the *probability of packet intercept* for an all-peer-to-peer workload, several hundred packets will be sent for each of the 3 peer-to-peer packet types.

The configurations and workloads for this experiment are given in Table 5.2. For the *packet processing time* test, all six workload factor levels are used. For the *probability of packet intercept* (non-peer-to-peer workload) test, the "BT On List", "SIP INVITE On List", and "SIP BYE On List" packet types are used. For the *probability of packet intercept* (all-peer-to-peer workload) test, the "BT On List", "SIP INVITE On List", and "SIP BYE On List" packet types are used.

5.10 Analysis and Interpretation of Results

5.10.1 Experiment 1: Finding an Optimal Configuration. The analysis of this experiment consists of performing a series of one-variable statistical computations and two-variable comparison tests to prove or disprove the hypothesis that a system equipped for a single peer-to-peer protocol (in this case, BitTorrent) can be constructed and optimized such that a packet of interest is detected and recorded with at least 95% probability. In addition, by analyzing the results of the *probability of packet intercept* (all-peer-to-peer workload) test for each configuration, a figure of merit is calculated for the probability of successfully intercepting multiple sequential packets of interest on a high utilization network.

5.10.1.1 Calculating Packet Processing Time. For each combination of CUT configuration and workload, a one variable t-test is performed to determine the mean *packet processing time* in CPU cycles, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean. Then, for each workload, the mean *packet processing time* of each CUT configuration is compared to the *packet processing time* of the Control configuration, and the reasons for any increase or decrease in processing time is analyzed.

5.10.1.2 Calculating Probability of Intercept Under Non-Peer-to-Peer and All-Peer-to-Peer Loads. For the configuration and workload combinations outlined in Section 5.9.1, a one proportion confidence interval analysis is performed on the binomial variable to determine the *probability of packet intercept* and a 95% confidence interval for the proportion. This generates a series of basic statistics from which to perform the two proportion hypothesis testing.

Next, a one-sided statistical hypothesis test using two proportions and a 95% confidence interval is performed for each of the five modification CUT configurations against the Control configuration. The results of these hypothesis tests determine which of the modification CUT configurations show statistically significant improvement over the Control configuration.

Finally, a one-sided statistical hypothesis test using two proportions and a 95% confidence interval is performed for the Combined configuration against the other four modification CUT configurations and against the Wireshark software-based packet sniffer. The results of these hypothesis tests determine if the improvement of the combined software configuration over each individual modification is statistically significant, and also determine if the hardware-based SUT is at least as effective as the software-based Wireshark system in intercepting single BitTorrent packets in a heavy non-peer-to-peer traffic environment and back-to-back BitTorrent packets of interest.

5.10.2 Experiment 2: Expanding the System. The analysis of this experiment consists of performing a series of one-variable statistical computations and two-variable comparison tests to prove or disprove the hypothesis that the system can be expanded to include a second peer-to-peer protocol (in this case, SIP) while maintaining at least a 95% probability of intercept for a packet of interest from either peer-to-peer protocol. By analyzing the results of the *probability of packet intercept* (all-peer-to-peer workload) test for each workload, a figure of merit is calculated for the probability of successfully intercepting multiple sequential packets of interest on a high utilization network.

5.10.2.1 Calculating Packet Processing Time. For each combination of CUT configuration and workload, a one variable t-test is performed to determine the mean *packet processing time* in CPU cycles, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean. Then, for the "Non-P2P", "BT On List", and "BT Off List" packet types, the mean *packet processing time* of the Optimized (BT + SIP) configuration is compared to the *packet processing time* of the Combined configuration from Experiment 1, and the reasons for any increase or decrease in processing time are analyzed.

5.10.2.2 Calculating Probability of Intercept Under Non-Peer-to-Peer and All-Peer-to-Peer Loads. For the configuration and workload combinations outlined in Section 5.9.2, a one proportion confidence interval analysis is performed on the binomial variable to determine the *probability of packet intercept* and a 95% confidence interval for the proportion. This generates a series of basic statistics from which to perform the two proportion hypothesis testing.

Then, a one-sided statistical hypothesis test using two proportions and a 95% confidence interval is performed for the Optimized (BT + SIP) configuration from Experiment 2 against the Combined configuration from Experiment 1. The result of this hypothesis test determines if there is any statistical difference in the *probability of packet intercept* of single BitTorrent packets in a heavy non-peer-to-peer traffic

environment and back-to-back BitTorrent packets of interest, when the system is expanded to included SIP functionality.

5.11 Summary

This chapter discusses the methodology used to evaluate the performance of the digital forensic tool under various workloads and network utilization scenarios. Performance is evaluated using a real-world experimental design and is based on two performance metrics: *packet processing time* and *probability of packet intercept*. Two partial-factorial experiments using three different tests are performed to measure the impact of varying the software configuration and the peer-to-peer input workload on overall system performance. An analysis is then performed on the data through a series of statistical tests to determine the effectiveness of the system using various configurations and workloads.

VI. Results and Analysis

This chapter presents and analyzes the experimental results. First, the results for each of the performance metrics for the three tests used in Experiment 1 are discussed in Section 6.1. Next, the results of the performance metrics for the three tests in Experiment 2 are presented in Section 6.2. Finally, an overall analysis of the results is given in Section 6.3.

6.1 Results and Analysis of Experiment 1

6.1.1 Test 1: Calculating Packet Processing Time. The first test performed on the system is used to determine how many CPU cycles are required to process each type of packet. Outlined below are the results of the Calculating Packet Processing Time test for each of the three packet type factors listed in Table 5.1.

6.1.1.1 Non-Peer-to-Peer Packet Workload. Table 6.1 shows the results of a one-variable t-test performed on each of the six configurations using the Non-Peer-to-Peer (labeled "Non-P2P" in Table 5.1) packet type. The table gives the number of trials, the mean number of CPU cycles required to process the non-peer-to-peer packet, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

Table 6.1: Packet Processing Times for Non-BitTorrent Packets

Configuration	N (Events)	Mean	Standard Deviation	Standard Error of the Mean	Confidence Interval (95%)
Control	50	1206.00	0.00	0.00	(1206.00, 1206.00)
User Alerts	50	1152.00	0.00	0.00	(1152.00, 1152.00)
Dual Buffer	50	1344.00	109.10	15.40	(1313.00, 1375.00)
Packet Write	50	1146.00	0.00	0.00	(1146.00, 1146.00)
Cache	50	276.00	0.00	0.00	(276.00, 276.00)
Combined	50	303.50	25.76	3.64	(296.18, 310.82)

Looking at the table, the initial assumption would be that only two of the six configurations, Dual Buffer and Combined, are stochastic processes. The other four, Control, User Alerts, Packet Write, and Cache, are all deterministic, as shown by

the null values for standard deviation and standard error of the mean. However, looking closer at the raw data, contained in Appendix B, the *packet processing times* in the Dual Buffer and Combined configurations in fact alternate between two values, which shows that these too, are deterministic processes. Thus, when processing non-peer-to-peer packets, every configuration results in a deterministic *packet processing time*.

Figure 6.1 shows 95% confidence interval plots of the *packet processing time* required for a packet that does not belong to the BitTorrent protocol. The number of cycles required ranges from 276 cycles to 1,344 cycles, which equates to a range of 0.92 to 4.48 microseconds per packet.

Looking at both the table and the figure, the following qualitative observations are noted:

- Enabling the User Alerts has no negative impact on *packet processing time* over the Control configuration. This is due to the fact that no alerts are generated when a non-peer-to-peer packet is processed.
- Adding a second receive buffer adds additional processing time to the Control configuration baseline. This is due to the additional software processing required to process packets from two receive buffers instead of one.
- Employing the alternate packet writing scheme has very little positive impact (less than 5%) on *packet processing time* over the Control configuration. This is due to the fact that nothing is written to the log file when a non-peer-to-peer packet is processed.
- A significant number of cycles are saved by enabling the instruction and data caches.
- The *packet processing time* for the Combined configuration is greater than that of the Cache configuration due to the Combined configuration incorporating the Dual Buffer Optimization, while the Cache configuration employs a single receive buffer.

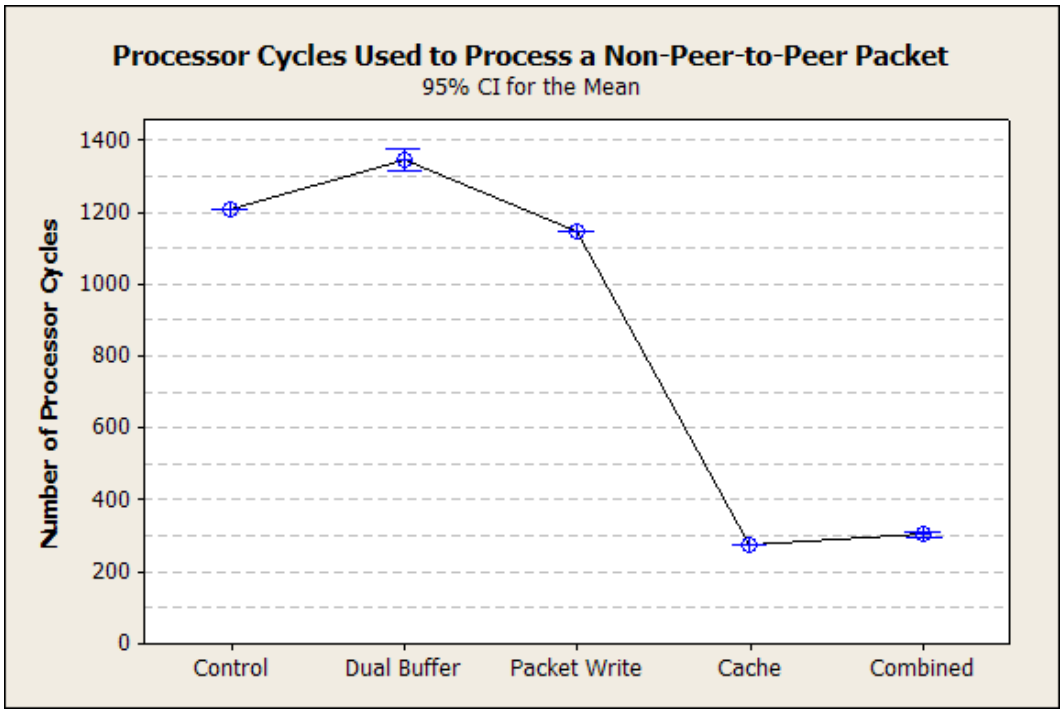

Figure 6.1: Interval Plots of Packet Processing Times for Non-BitTorrent Packets

6.1.1.2 BitTorrent Packet Not On the List Workload. Table 6.2 shows the results of a one-variable t-test performed on each of the six configurations using the BitTorrent Packet Not On the List (labeled "BT Off List" in Table 5.1) packet type. The table gives the number of trials, the mean number of CPU cycles required to process the BitTorrent Handshake packet, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

Table 6.2: Packet Processing Times for BitTorrent Packets Not On the List

Configuration	N (Events)	Mean	Standard Deviation	Standard Error of the Mean	Confidence Interval (95%)
Control	50	7296.00	0.00	0.00	(7296.00, 7296.00)
User Alerts	50	1044756	730	103	(1044549, 1044963)
Dual Buffer	50	7770.00	0.00	0.00	(7770.00, 7770.00)
Packet Write	50	7593.00	0.00	0.00	(7593.00, 7593.00)
Cache	50	1145.00	0.00	0.00	(1145.00, 1145.00)
Combined	50	1205.00	0.00	0.00	(1205.00, 1205.00)

Looking at the table and the raw data, only one of the six configurations, User Alerts, is a stochastic process. The other five, Control, Dual Buffer, Packet Write, Cache, and Combined, are all deterministic, as shown by the null values for standard deviation and standard error of the mean. Thus, for BitTorrent traffic containing file info hashes that are not on the list of interest, the only configuration that does not result in a deterministic *packet processing time* is the User Alert configuration. The primary reason for the stochastic nature of the User Alert Configuration is that information sent to the user via RS232 port is subject to processor bus contention issues on the FPGA board, resulting in a variable transmission time.

Figure 6.2 shows 95% confidence interval plots of the *packet processing time* required for a packet that does not belong to the BitTorrent protocol. The number of cycles required ranges from 1,145 cycles to 7,770 cycles, which equates to a range of 3.82 to 25.90 microseconds per packet.

Looking at both the table and the figure, the following qualitative observations are noted:

- Enabling the User Alerts results in a significant increase in *packet processing time*, due to the slow rate of RS232 data transmission compared to CPU speed. Because the *packet processing time* penalty for enabling User Alerts is so high, for clarity purposes, the User Alerts configuration does not appear in the figure.
- Adding a second receive buffer adds additional processing time to the Control configuration baseline. This is due to the additional software processing required to process packets from two receive buffers instead of one.
- Employing the alternate packet writing scheme has very little negative impact (less than 5%) on *packet processing time* over the Control configuration. This is due to the fact that nothing is written to the log file when a BitTorrent packet that is not of interest is processed.
- A significant number of cycles are saved by enabling the instruction and data caches.

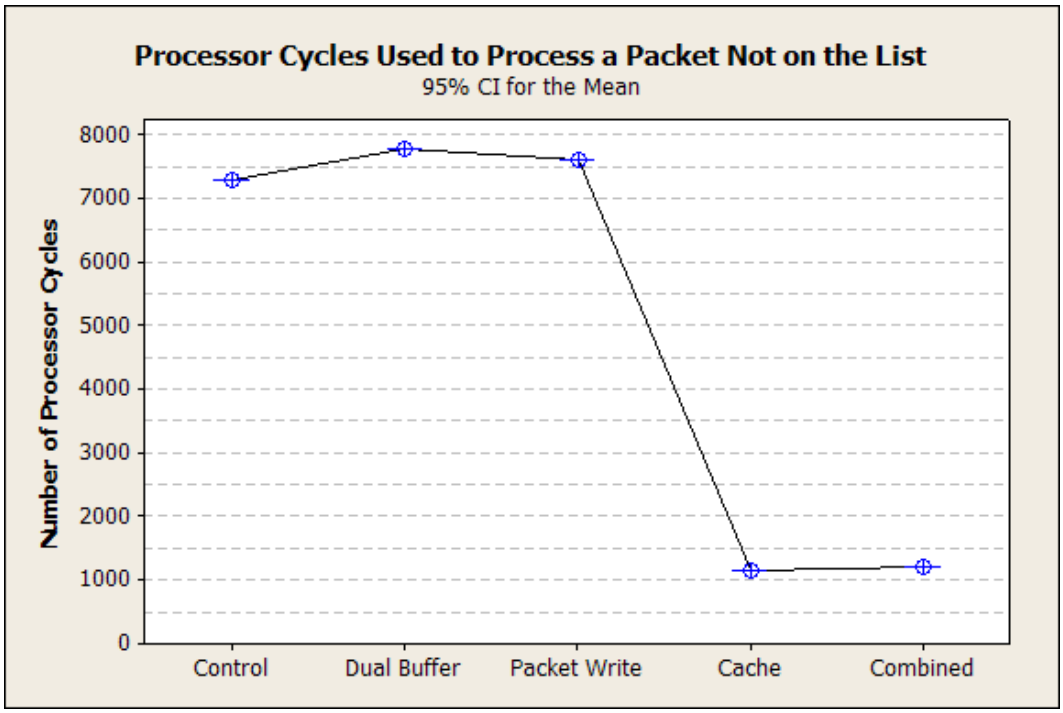

Figure 6.2: Interval Plots of Packet Processing Times for BitTorrent Packets Not On the List

- The *packet processing time* for the Combined configuration is greater than that of the Cache configuration due to the Combined configuration incorporating the Dual Buffer Optimization, while the Cache configuration employs a single receive buffer.

6.1.1.3 BitTorrent Packet On the List Workload. Table 6.3 shows the results of a one-variable t-test performed on each of the six configurations using the BitTorrent Packet On the List (labeled "BT On List" in Table 5.1) packet type. The table gives the number of trials, the mean number of CPU cycles required to process the BitTorrent Handshake packet, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

Looking at the table, the initial assumption would be that all six configurations are stochastic processes. However, looking closer at the raw data, contained in Appendix B, the *packet processing time* in the Packet Write configuration in fact

Table 6.3: Packet Processing Times for BitTorrent Packets On the List

Configuration	N (Events)	Mean	Standard Deviation	Standard Error of the Mean	Confidence Interval (95%)
Control	50	116207	22418	3170	(109836, 122578)
User Alerts	50	1702125	22880	3236	(1695623, 1708628)
Dual Buffer	50	118986	22391	3167	(112623, 125350)
Packet Write	50	23292	318	45	(23202, 23382)
Cache	50	14679	2064	292	(14093, 15266)
Combined	50	3783	75	11	(3762, 3805)

alternates between two values, which shows that this is actually a deterministic process. Thus, when BitTorrent packets are processed and then written to a log file, every configuration save one results in a variable *packet processing time*.

Figure 6.3 shows 95% confidence interval plots of the *packet processing time* required for a BitTorrent Handshake packet whose file info hash is on the list of interest. The number of cycles required ranges from 3,783 cycles to 118,986 cycles, which equates to a range of 12.61 to 396.62 microseconds per packet.

Looking at both the table and the figure, the following qualitative observations are noted:

- Enabling the User Alerts results in a significant increase in *packet processing time*, due to the slow rate of RS232 data transmission versus CPU speed. Again, for clarity purposes, the User Alerts configuration does not appear in the figure.
- Adding a second receive buffer adds a small amount of additional processing time to the Control configuration baseline. This is due to the additional software processing required process frames from two receive buffers instead of one.
- Writing the frame to on-board RAM instead of the Compact Flash card, as is done in the Packet Write optimization, reduces the *packet processing time* by approximately 80% over the Control configuration.
- A significant number of cycles are saved by enabling the instruction and data caches.

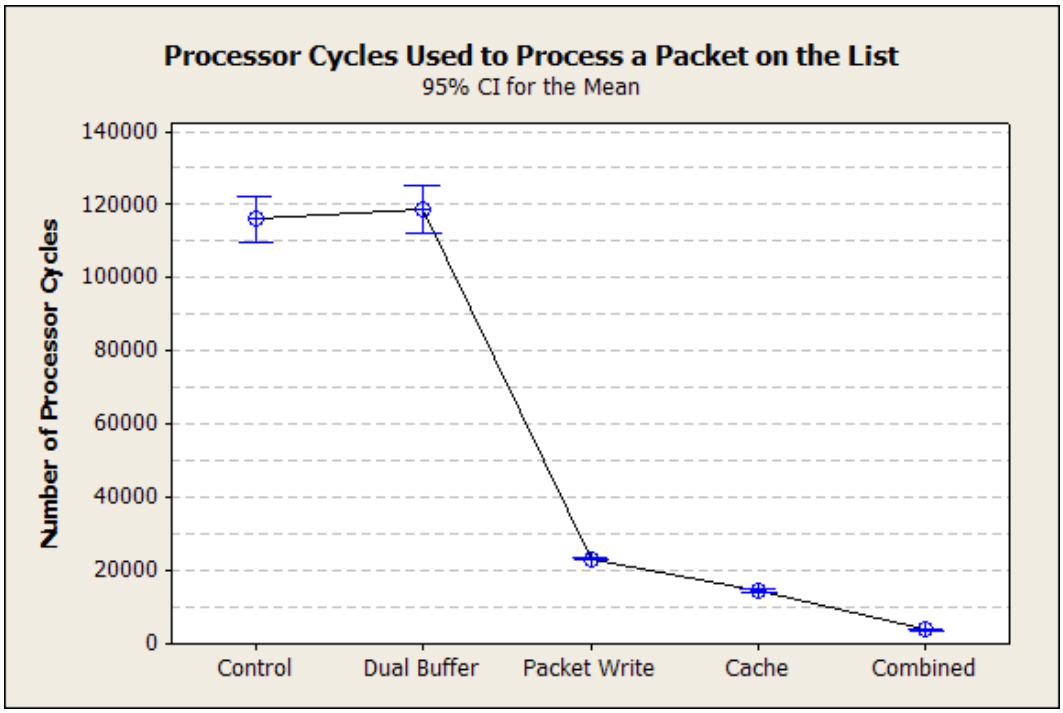

Figure 6.3: Interval Plots of Packet Processing Times for BitTorrent Packets On the List

- The *packet processing time* for the Combined configuration is now much less than that of the Cache configuration. This is due to the large *packet processing time* reduction of the Packet Write and Cache optimizations outweighing the small *packet processing time* increase of the Dual Buffer optimization.

6.1.2 Test 2: Calculating Probability of Intercept Under a Non-Peer-to-Peer Load. Table 6.4 shows the results of the packet intercept test under a heavy non-peer-to-peer network load. For each configuration tested, the number of packets captured out of the 300 sent is shown. The table also shows the *probability of packet intercept* and the corresponding 95% confidence interval for each configuration. In all tests, the total load on the network is measured by the Wireshark packet sniffer to be between 89.6 Mbps and 89.7 Mbps, which equates to an 89.6% load on the 100 Mbps network. However, this measurement is not absolute, as the Wireshark program itself can drop packets under a heavy load. Since it is unknown how many packets

were dropped by Wireshark, 89.6% is considered to be the minimum load on the test network.

Table 6.4: Probability of Packet Intercept Under a Non-Peer-to-Peer Workload

Configuration	Packets Captured (Events)	Packets Sent (Trials)	Probability of Packet Intercept	Confidence Interval (95%)
Control	159	300	0.5300	(0.4718, 0.5876)
User Alerts	166	300	0.5533	(0.4951, 0.6105)
Dual Buffer	292	300	0.9733	(0.9481, 0.9884)
Packet Write	174	300	0.5800	(0.5219, 0.6365)
Cache	289	300	0.9633	(0.9353, 0.9816)
Combined	300	300	1.0000	(0.9901, 1.0000)
Wireshark	298	300	0.9933	(0.9761, 0.9992)

Figure 6.4 shows the 95% confidence intervals of *probability of packet intercept* for the configurations in Table 6.4. Looking at the table and the figure, while the User Alerts and Packet Write configurations capture more packets of interest than the Control (166 and 174 versus 159), the overlapping confidence intervals shown in the table and figure suggest that the differences are not statistically significant. The figure and table also show that the Cache and Dual Buffer configurations perform significantly better than the Control. Moreover, the Combined configuration performs better than the other five configurations (300 out of 300 packets captured), returning a test result of 100% *probability of packet intercept* for packets of interest, which is comparable to the 99% capture rate of the Wireshark packet sniffer.

To further determine the statistical significance of these results, hypothesis tests are performed between the various optimizations versus the Control configuration. As shown in Table 6.5, the p-value for the one-sided test involving the User Alerts and the Control is too high (0.283) to state with confidence that the increase in *probability of packet intercept* is statistically significant. In the one-sided test involving the Packet Write optimization and the Control, again the p-value is too high (0.109) to accept the alternative hypothesis, but it can be inferred that the optimization did provide some improvement to the *probability of packet intercept*. For the Cache, Dual Buffer,

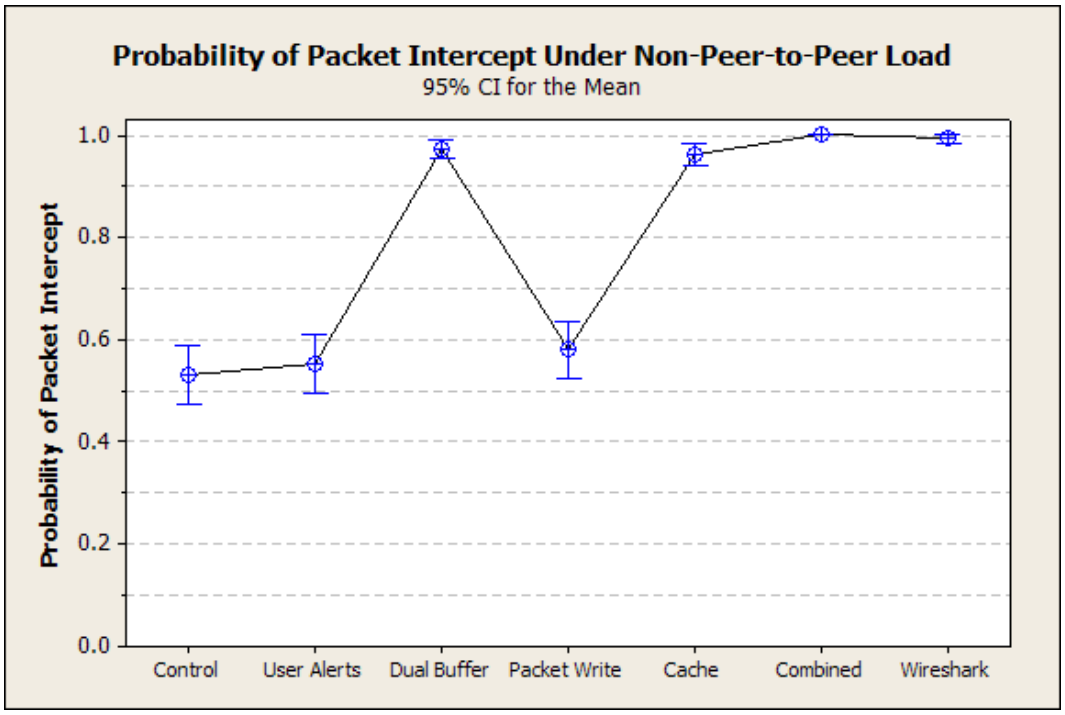

Figure 6.4: Interval Plots of Probability of Intercept for a BitTorrent Packet Under a Non-Peer-to-Peer Workload

and Combined configurations, the p-value for the one-sided test is 0.000, indicating a strong statistical certainty that these configurations are better than the Control configuration.

Table 6.5: Hypothesis Testing on Control Configuration Under a Non-Peer-to-Peer Workload

Alternative Hypothesis with 95% Confidence Interval	Estimate for Difference	Z Value of Difference Test	P Value of Difference Test
$p(User\ Alerts) > p(Control)$	0.0233	0.57	0.283
$p(Dual\ Buffer) > p(Control)$	0.4433	14.64	0.000
$p(Packet\ Write) > p(Control)$	0.0500	1.23	0.109
$p(Cache) > p(Control)$	0.4333	14.07	0.000
$p(Combined) > p(Control)$	0.4700	16.31	0.000

To determine the overall performance of the Combined configuration, another set of hypothesis tests are performed between the Combined configuration versus the individual optimizations and Wireshark. As shown in Table 6.6, the p-value for the one-sided tests involving the User Alerts, Packet Write, Cache, and Dual Buffer

optimizations ranges between 0.000 and 0.002, indicating a strong statistical certainty that the Combined configuration is better than each individual optimization by itself. When the Combined configuration is compared to the performance of Wireshark, the p-value for the one-sided test is 0.078, which is too high to accept the alternative hypothesis that the Combined configuration performs better than Wireshark, but does indicate that the *probabilities of packet intercept* of the two are comparable.

Table 6.6: Hypothesis Testing on Combined Configuration Under a Non-Peer-to-Peer Workload

Alternative Hypothesis with 95% Confidence Interval	Estimate for Difference	Z Value of Difference Test	P Value of Difference Test
$p(Combined) > p(User\ Alerts)$	0.4467	15.56	0.000
$p(Combined) > p(Dual\ Buffer)$	0.0267	2.87	0.002
$p(Combined) > p(Packet\ Write)$	0.4200	14.74	0.000
$p(Combined) > p(Cache)$	0.0367	3.38	0.000
$p(Combined) > p(Wireshark)$	0.0067	1.42	0.078

6.1.3 Test 3: Calculating Probability of Intercept Under an All-Peer-to-Peer Load. Table 6.7 shows the results of the packet intercept test under an all-peer-to-peer network load. For each configuration tested, the number of BitTorrent Handshake packets that are sent over the network in order for the system to capture 400 of them is shown in the table. The table also shows the *probability of packet intercept* and the corresponding 95% confidence interval for each configuration. In all tests, the total load on the network is measured by the Wireshark packet sniffer to be between 23.35 and 24.10 Mbps, which equates to approximately a 23.3% load on the 100 Mbps network. This is the maximum network throughput possible using the Hping utility and the BitTorrent Handshake workload.

Figure 6.5 shows the 95% confidence intervals of probability of capture for the configurations in Table 6.7. Looking at the table and the figure, the only configuration that performs worse than the Control is the User Alerts configuration (3.4% capture rate for Control versus 1.5% capture rate for User Alerts). The figure and table also show that the Cache and Dual Buffer configurations perform slightly better than the

Table 6.7: Probability of Packet Intercept Under an All-Peer-to-Peer Workload

Configuration	Packets Captured (Events)	Packets Sent (Trials)	Probability of Packet Intercept	Confidence Interval (95%)
Control	400	11757	0.0340	(0.0308, 0.0375)
User Alerts	400	26810	0.0149	(0.0135, 0.0164)
Dual Buffer	400	9188	0.0435	(0.0395, 0.0479)
Packet Write	400	990	0.4040	(0.3733, 0.4354)
Cache	400	3599	0.1111	(0.1011, 0.1219)
Combined	400	400	1.0000	(0.9925, 1.0000)
Wireshark	400	404	0.9901	(0.9748, 0.9973)

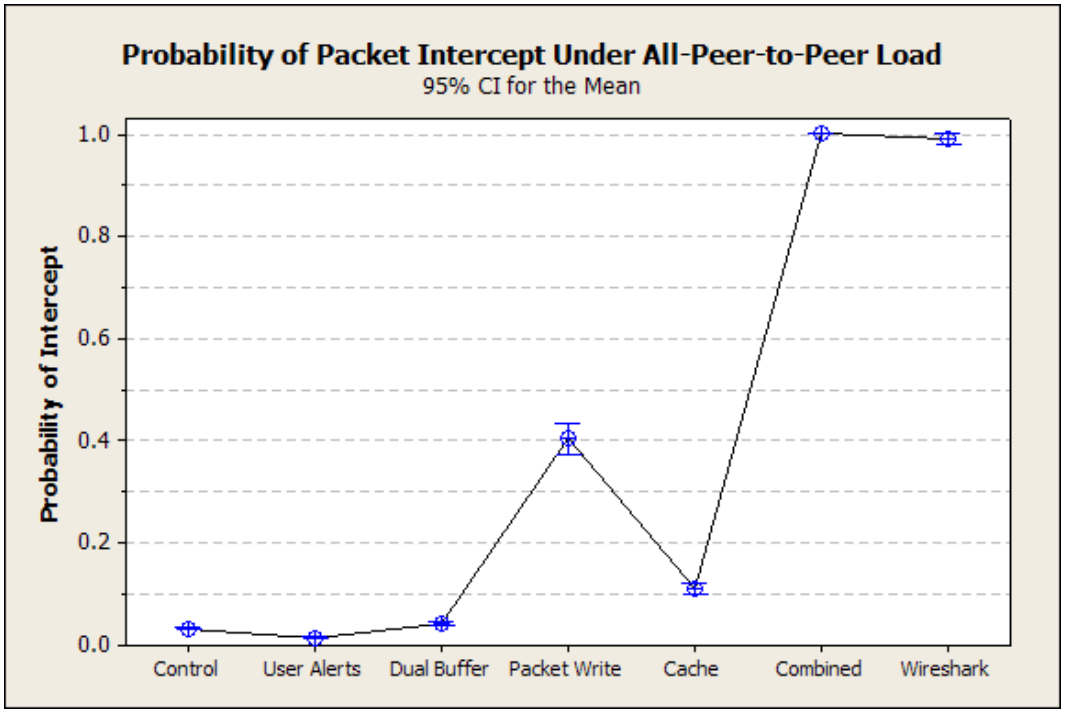

Figure 6.5: Interval Plots of Probability of Intercept for a BitTorrent Packet Under an All-Peer-to-Peer Workload

Control, but still are only able to capture less than 12% of packets sent. The Packet Write configuration performs moderately better than the Control (40.4% versus 3.4% capture rate), but it is still unable to capture more than 1 in 2 packets. Finally, the Combined configuration performs significantly better than the other five configurations (400 out of 400 packets captured), returning a test result of 100% *probability of packet intercept* and comparing very favorably with Wireshark's result of 99.0%.

To further validate the statistical significance of these results, hypothesis tests are performed between the various optimizations versus the Control configuration. As shown in Table 6.8, the p-value for the one-sided test involving the User Alerts and the Control is 1.000, which corresponds to the fact that the User Alerts configuration actually performed worse than the Control. In the one-sided tests involving the other four optimizations, the p-value for the one-sided test is 0.000, indicating a strong statistical certainty that these configurations have a better *probability of packet intercept* than the Control configuration.

Table 6.8: Hypothesis Testing on Control Configuration Under an All-Peer-to-Peer Workload

Alternative Hypothesis with 95% Confidence Interval	Estimate for Difference	Z Value of Difference Test	P Value of Difference Test
$p(User\ Alerts) > p(Control)$	(0.0191)	(10.45)	1.000
$p(Dual\ Buffer) > p(Control)$	0.0095	3.51	0.000
$p(Packet\,Write) > p(Control)$	0.3700	23.59	0.000
$p(Cache) > p(Control)$	0.0771	14.02	0.000
$p(Combined) > p(Control)$	0.9660	577.76	0.000

To determine the overall performance of the Combined configuration, another set of hypothesis tests are performed between the Combined configuration versus the individual optimizations and Wireshark. As shown in Table 6.9, the p-value for the one-sided tests involving the User Alerts, Dual Buffer, Packet Write, and Cache optimizations are all 0.000, indicating a strong statistical certainty that the Combined configuration is better than each individual optimization by itself. When the Combined configuration is compared to the performance of Wireshark, the p-value for the one-sided test is 0.022, which is low enough to accept, with statistical confidence, the hypothesis that *probability of packet intercept* for the Combined configuration is higher than the *probability of packet intercept* using Wireshark.

6.1.4 Experiment 1 Analysis. Table 6.10 shows a summary of the results from the first test, Calculating Packet Processing Time. For each workload, the mean

Table 6.9: Hypothesis Testing on Combined Configuration Under an All-Peer-to-Peer Workload

Alternative Hypothesis with 95% Confidence Interval	Estimate for Difference	Z Value of Difference Test	P Value of Difference Test
$p(Combined) > p(User\ Alerts)$	0.9851	1330.46	0.000
$p(Combined) > p(Dual\ Buffer)$	0.9565	449.29	0.000
$p(Combined) > p(Packet\ Write)$	0.5960	38.21	0.000
$p(Combined) > p(Cache)$	0.8889	169.66	0.000
$p(Combined) > p(Wireshark)$	0.0099	2.01	0.022

packet processing times and the percentage change from the Control configuration are recorded in the table.

Table 6.10: Mean Packet Processing Time Comparisons to Control Configuration

Configuration	Non-BT Packet (Cycles)	Percent Change (%)	BT Packet Not on List (Cycles)	Percent Change (%)	BT Packet on List (Cycles)	Percent Change (%)
Control	1206	0.00	7296	0.00	116207	0.00
User Alerts	1152	4.48	1044756	(14219.60)	1702125	(1364.74)
Dual Buffer	1344	(11.44)	7770	(6.50)	118986	(2.39)
Packet Write	1146	4.98	7593	(4.07)	23292	79.96
Cache	276	77.11	1145	84.31	14679	87.37
Combined	304	74.83	1205	83.48	9125	92.15

Analyzing the table, the following observations are made about the data collected in the first test:

- For the two BitTorrent packet workloads, messages to the user were sent in the User Alert configuration, resulting in a several order of magnitude increase in processing time. This increase is due to the fact that the user alerts are transmitted via serial port at 115,200 baud, which is significantly slower than the 300 MHz processor speed and 100 MHz bus speed used by the FPGA board. Based on this significant increase in *packet processing time*, and the corresponding decrease in overall system performance, all user alerts are eliminated from the final design.

- Adding a second receive buffer results in more CPU cycles required to process a packet, regardless of the type of packet. This is due to the additional processing cycles required to check both receive buffers in order to determine which one contains the next packet to be processed. However, as shown in the second test, this increase in CPU cycles is more than offset by the advantages gained by implementing the second Ethernet receive buffer.
- As expected, the modification to the packet writing routine only decreases the *packet processing time* when packets are actually written to the log file. For the cases where packets are not written, no significant processing time is gained or lost with this optimization.
- Enabling the caches results in a significant decrease in CPU cycles required to process a packet, regardless of packet type.

Table 6.11 shows a summary the results from the second test, Calculating Probability of Intercept Under a Non-Peer-to-Peer Load. For each configuration, the measured *probability of packet intercept* and the percentage change from the Control configuration are recorded in the table. Analyzing the data in this table, combined with the data in Tables 6.4, 6.5, and 6.6, the following observations are made about the data collected in the second test:

Table 6.11: Comparison of Probability of Packet Intercept Between Optimizations and Control Configuration for a Non-Peer-to-Peer Workload

Configuration	Probability of Packet Intercept	Percent Change from Control (%)
Control	0.5300	0.00
User Alerts	0.5533	4.40
Dual Buffer	0.9733	83.64
Packet Write	0.5800	9.43
Cache	0.9633	81.76
Combined	1.0000	88.68
Wireshark	0.9933	79.52

- Adding User Alerts to the system has no statistical impact on the *probability of packet intercept*, positive or negative. However, given the vast increase in *packet processing time* associated with messages sent to the user (5,673.8 microseconds with the User Alerts versus 387.4 without User Alerts), their removal is still justified in the final system design. This point is discussed further in Section 6.3.
- The alternate Packet Writing scheme does not, by itself, significantly improve overall system performance. However, this optimization, when combined with other improvements, does provide some benefit to the Combined configuration's performance.
- The optimizations that enable the caches and the second receive buffer each have a significant positive impact on overall system performance. For each optimization, system performance increased over 80% from the Control configuration.
- The combination of all three optimizations returned the best performance of any configuration. Even with the additional processing required to analyze all packets on the network for the BitTorrent protocol signature, the system returned similar performance to the dedicated software-based packet sniffer, Wireshark.

Table 6.12 shows a summary the results from the third test, Calculating Probability of Intercept Under an All-Peer-to-Peer Load. For each configuration, the measured *probability of packet intercept* and the percentage change from the Control configuration are recorded in the table. Analyzing the data in this table, combined with the data in Tables 6.7, 6.8, and 6.9, the following observations are made about the data collected in the third test:

- Adding User Alerts to the system results in a 56% decrease in performance, as measured by *probability of packet intercept*. In this case, the vast increase in *packet processing time* associated with messages sent to the user, discussed above, is almost certainly the root cause of the decrease in performance.

Table 6.12: Comparison of Probability of Packet Intercept Between Optimizations and Control Configuration for an All-Peer-to-Peer Workload

Configuration	Probability of Packet Intercept	Percent Change from Control (%)
Control	0.0340	0.00
User Alerts	0.0149	(56.18)
Dual Buffer	0.0435	27.94
Packet Write	0.4040	1088.24
Cache	0.1111	226.76
Combined	1.0000	2841.18
Wireshark	0.9901	2812.06

- The Dual Buffer and Cache optimizations, by themselves, modestly improve system performance, but are still unable to capture more than 50% of packets of interest. However, combining them with the Packet Write optimization provides a tremendous benefit to the Combined configuration's performance.
- The alternate Packet Writing scheme by itself provides a moderate improvement to overall system performance. The full benefit of this optimization is seen when combined with caching and an improved Ethernet receive buffer.
- The combination of all three optimizations (Cache, Dual Buffer, and Packet Write) have a synergistic effect on the overall performance of the system when processing back-to-back BitTorrent packets. By themselves, each optimization returned moderate performance gains over the Control configuration. When combined, however, they created a system that is able to achieve a 100% *probability of packet intercept*, which is comparable to the dedicated software packet sniffer, Wireshark.

Overall, the Combined configuration is clearly the best of the possible configurations for the CUT. The Combined configuration consistently returned very low *packet processing times*, indicating that it is able to process a variety of packets faster than any individual optimization. The Combined configuration also returned the highest values in both *probability of packet intercept* tests, indicating that it has a higher

probability of intercepting packets of interest in both non-peer-to-peer and all-peer-to-peer workload environments than any of the other optimizations. Finally, using a 95% confidence interval, the Combined configuration returns a minimum capture rate of 99.0% across all workloads, which is comparable to the performance of Wireshark, which returned a minimum capture rate of 97.5%.

6.2 Results and Analysis of Experiment 2

6.2.1 Test 1: Calculating Packet Processing Time. Table 6.13 shows the results of a one-variable t-test performed on the Optimized (BT + SIP) configuration using six different packet types. The table gives the number of trials, the mean number of CPU cycles required to process the workload packet, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

Table 6.13: Packet Processing Times for SIP and BitTorrent Packets Using the Expanded System

Configuration	N (Events)	Mean	Standard Deviation	Standard Error of the Mean	Confidence Interval (95%)
Not P2P	50	418.6	36.0	5.1	(408.4, 428.4)
BT Not on List	50	1323.5	31.8	4.5	(1314.5, 1332.5)
BT Handshake	50	3883.0	85.6	12.1	(3858.7, 3907.4)
SIP Not on List	50	19450.0	97.8	13.8	(19422.2, 19477.8)
SIP BYE	50	29951.3	224.2	31.7	(29887.6, 30015.0)
SIP INVITE	50	34778.6	226.2	32.0	(34714.3, 34842.9)

Looking at the table, the *packet processing times* for all six packet types are initially assumed to be stochastic processes. However, looking closer at the raw data, contained in Appendix B, the *packet processing time* for the "Non-P2P" packet type alternates between two values, and the *packet processing time* for the "BT Off List" packet type returns the same value for every packet except the first. These results match those of the first experiment in that the *packet processing times* are deterministic for "Non-P2P" and "BT Off List" packet types in both experiments.

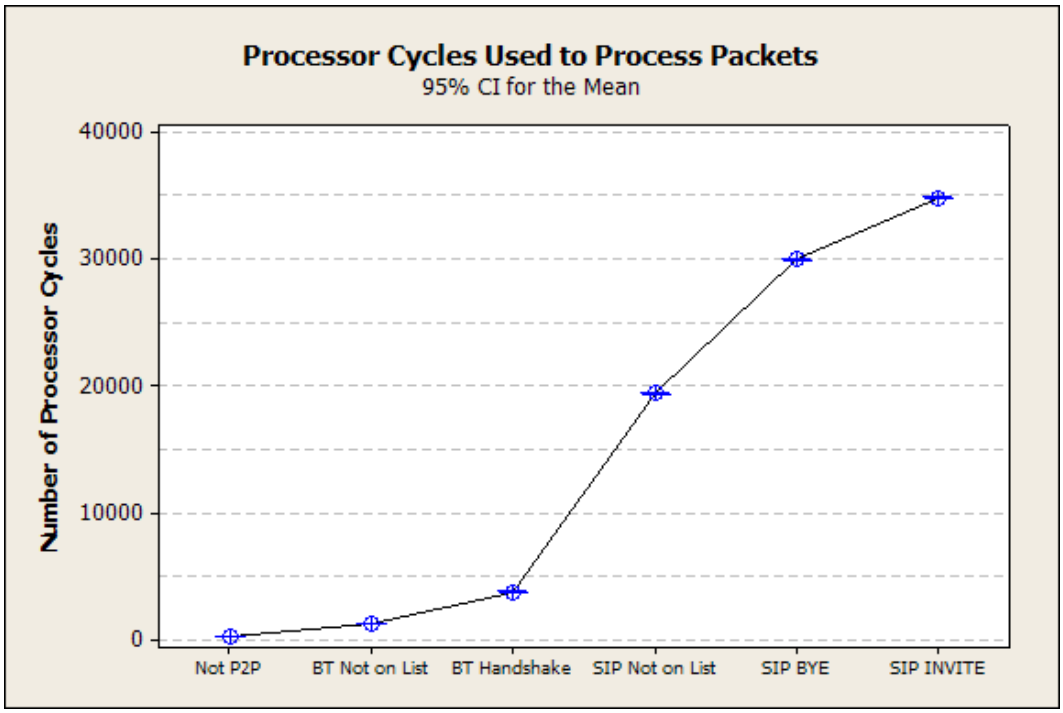

Figure 6.6: Interval Plots of Packet Processing Times for SIP and BitTorrent Packets

Figure 6.6 shows 95% confidence interval plots of the *packet processing times* required for each of the six packet types. The number of cycles required ranges from 419 cycles to 34,779 cycles, which equates to a range of 1.40 to 115.93 microseconds per packet, depending on the type of packet.

To compare the original system's performance from the first experiment against the system's performance with the addition of SIP protocol processing capability, hypothesis tests are performed between the Combined configuration from Experiment 1 against the modified Optimized (BT + SIP) configuration from Experiment 2. As shown in Table 6.14, the p-value for the one-sided tests involving the *packet processing times* for the "Non-P2P" and "BT On List" packet types is 0.000, which implies that the addition of SIP protocol processing capability results in increased *packet processing times* for both packet types over the BitTorrent-only system.

Table 6.14: Hypothesis Testing on Expanded System (BitTorrent+SIP) versus the BitTorrent-Only System

Workload	Alternative Hypothesis with 95% Confidence Interval	Estimate for Difference	T Value of Difference Test	P Value of Difference Test
Non-P2P	$(BT + SIP) > (Combined)$	115.12	18.38	0.000
BT On List	$(BT + SIP) > (Combined)$	99.80	6.22	0.000

6.2.2 Test 2: Calculating Probability of Intercept Under a Non-Peer-to-Peer Load. Table 6.15 shows the results of the packet intercept test under a heavy non-peer-to-peer network load using the modified Optimized (BT + SIP) configuration. For each of the three peer-to-peer packet types tested, the number of packets captured by the system out of the 300 sent is shown. For comparison purposes, the number of packets captured by the Wireshark packet sniffer for each workload is also shown. In addition, the table shows the *probability of packet intercept* and the corresponding 95% confidence interval for each workload. In all tests, the total load on the network is measured by the Wireshark packet sniffer to be between 89.6 Mbps and 89.9 Mbps, which equates to a minimum 89.6% load on the 100 Mbps network, matching the results from Experiment 1.

Table 6.15: Probability of Packet Intercept for BitTorrent and SIP Packets Under a Non-Peer-to-Peer Workload

Workload	Packets Captured (Events)	Packets Sent (Trials)	Probability of Packet Intercept	Confidence Interval (95%)
BT Handshake	300	300	1.0000	(0.9901, 1.0000)
Wireshark BT	298	300	0.9933	(0.9761, 0.9992)
SIP BYE	300	300	1.0000	(0.9901, 1.0000)
Wireshark BYE	300	300	1.0000	(0.9901, 1.0000)
SIP INVITE	298	300	0.9933	(0.9761, 0.9992)
Wireshark INVITE	300	300	1.0000	(0.9901, 1.0000)

Figure 6.7 shows the 95% confidence intervals of *probability of packet intercept* for the workloads in Table 6.15. Looking at the table and the figure, the modified Optimized (BT + SIP) configuration performed perfectly (300 out of 300 packets captured) for two out of the three workloads, and returned a 99% *probability of packet*

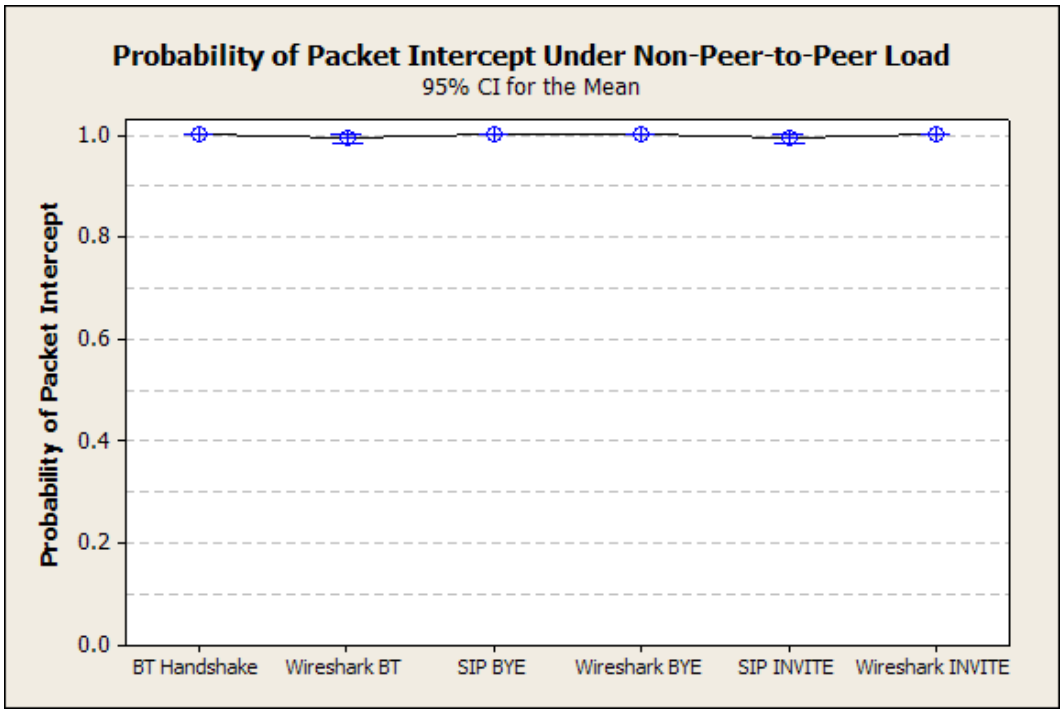

Figure 6.7: Interval Plots of Probability of Intercept for BitTorrent and SIP Packets Under a Non-Peer-to-Peer Workload

intercept for the other. This performance compares very favorably with the results returned by the Wireshark packet sniffer, which also returned a test result of near-100% *probability of packet intercept* for packets of interest.

6.2.3 Test 3: Calculating Probability of Intercept Under an All-Peer-to-Peer Load. For each of the three all-peer-to-peer workloads (BitTorrent Handshake, SIP INVITE, and SIP BYE), the total load on the network is measured by the Wireshark packet sniffer, and the results shown in Table 6.16. The low maximum network load for the BitTorrent Handshake packet workload is likely due to the fact that the BitTorrent peer wire protocol runs on top of TCP. Both the exponential backoff mechanism and the reliable data transfer features of TCP add additional time between packets sent over the network, causing a decrease in the maximum throughput that the Hping program can achieve. The SIP INVITE and BYE packets, on the other hand, use UDP, which allows Hping to achieve a throughput of over 94 Mbps on the 100 Mbps network.

Table 6.16: Observed Network Load for Various All-Peer-to-Peer Workloads

Configuration	Network Load (Mbps)
BitTorrent Handshake	23.35
SIP BYE	94.61
SIP INVITE	96.28

Table 6.17 shows the results of the packet intercept test under an all-peer-to-peer network load. For each peer-to-peer packet type tested, the number of workload packets that were sent over the network in order for the system to capture 400 of them is shown. For comparison purposes, the number of packets captured by the Wireshark packet sniffer for each workload is also shown. In addition, the table shows the *probability of packet intercept* and the corresponding 95% confidence interval for each configuration.

Table 6.17: Probability of Packet Intercept for BitTorrent and SIP Packets Under an All-Peer-to-Peer Workload

Configuration	Packets Captured (Events)	Packets Sent (Trials)	Probability of Packet Intercept	Confidence Interval (95%)
BT Handshake	400	400	1.0000	(0.9901, 1.0000)
Wireshark BT	400	404	0.9901	(0.9748, 0.9973)
SIP BYE	400	440	0.9091	(0.8783, 0.9343)
Wireshark BYE	400	400	1.0000	(0.9901, 1.0000)
SIP INVITE	400	608	0.6579	(0.6187, 0.6956)
Wireshark INVITE	400	401	0.9975	(0.9862, 0.9999)

Figure 6.8 shows the 95% confidence intervals of *probability of packet intercept* for the workloads in Table 6.17. Looking at the table and the figure, both the SUT and Wireshark perform very well under the BitTorrent Handshake packet type, returning a *probability of packet intercept* of over 99%. For the SIP BYE packet type, the SUT returns a *probability of packet intercept* of just over 90%, while Wireshark returns a perfect score of 100%. Finally, for the SIP INVITE packet type, the SUT achieves a 65.8% *probability of packet intercept*, while Wireshark performs much better, returning a near-perfect score of 99.8%.

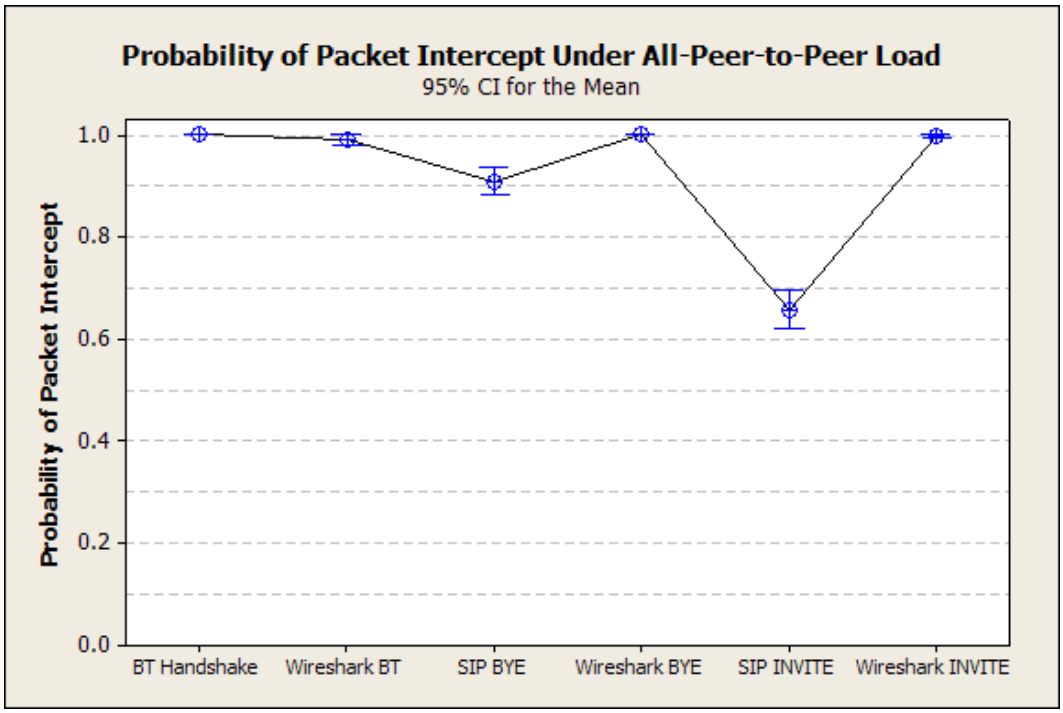

Figure 6.8: Interval Plots of Probability of Intercept for BitTorrent and SIP Packets Under an All-Peer-to-Peer Workload

6.2.4 Experiment 2 Analysis. Looking at Tables 6.13 and 6.14 the following observations are made about the data collected in the first test:

- Adding SIP processing capability to the SUT results in a higher *packet processing time* for both non-peer-to-peer and BitTorrent Handshake packets. This is due to the additional processing required by the system to determine if a packet belongs to either the BitTorrent or SIP protocols, as opposed to looking for only BitTorrent packets.
- The *packet processing time* required for any SIP packet is several times longer than the time required to process BitTorrent packets. The reasons for this increase in processing time are explained in Section 6.3.

Table 6.18 shows a summary the results from the second and third tests, Calculating Probability of Intercept Under a Non-Peer-to-Peer Load and Under an All-Peer-to-Peer Load. For each packet type, the measured *probability of packet intercept* for the non-peer-to-peer and all-peer-to-peer workloads and the percentage difference

between the two are recorded in the table. Analyzing the data in this table, combined with the data in Tables 6.15 and 6.17, the following observations are made about the data collected in the second and third tests:

Table 6.18: Comparison of Probability of Packet Intercept Between Non-Peer-to-Peer and All-Peer-to-Peer Workloads

Workload	Probability of Packet Intercept (Non-Peer-to-Peer)	Probability of Packet Intercept (All-Peer-to-Peer)	Improvement (%)
BT Handshake	1.0000	1.0000	0.00
Wireshark BT	0.9933	0.9901	(0.32)
SIP BYE	1.0000	0.9091	(9.09)
Wireshark BYE	1.0000	1.0000	0.00
SIP INVITE	0.9933	0.6579	(33.76)
Wireshark INVITE	1.0000	0.9975	(0.25)

- The *probability of packet intercept* performance of the system is unchanged when processing BitTorrent Handshake packets. Regardless of whether the SUT is processing a single BitTorrent Handshake packet amid a high non-peer-to-peer network load or a steady stream of Handshake packets, the system is able to achieve a 100% *probability of packet intercept*. In addition, the SUT performs slightly better than the Wireshark packet sniffer, regardless of the overall workload.
- When processing SIP BYE packets, the SUT sees a 9% decrease in *probability of packet intercept* when processing the packets back-to-back over processing a single packet amid a high non-peer-to-peer network load. The extended *packet processing time* required for this type of packet causes the system to occasionally drop the next frame entering the SUT because it is still processing the current SIP frame in the Ethernet receive buffer. The Wireshark packet sniffer, however, returns perfect scores regardless of workload type.
- When processing SIP INVITE packets, the SUT sees a 33.8% decrease in *probability of packet intercept* when processing the packets back-to-back over pro-

cessing a single packet amid a high non-peer-to-peer network load. The reason for this is the same as that for the SIP BYE packet. However, the SIP INVITE packet, due to its larger overall packet size, requires a longer *packet processing time* than the BYE packet, resulting in a lower *probability of packet intercept* than that of the SIP BYE packet. Wireshark, however, does not suffer from this problem, returning a *probability of packet intercept* of at least 99.7% for both workloads.

6.3 Overall Analysis

6.3.1 Analysis of Packet Processing Time. The first step in the research methodology is to find a system configuration that required the minimum number of CPU cycles to process packets entering the system. Based on the results presented here, the most significant improvement to system speed occurs when the data and instruction caches are enabled for the Power PC processor. By allowing the FPGA to cache both processor instructions and heap and stack data, packet processing time is reduced by 77% to 87%, depending on the type of packet. In addition, by delaying the Compact Flash write operations until after the termination of system processing, the *packet processing time* is reduced by 80% for packets written to the log file. When all four optimizations are combined, the resulting Combined configuration achieves a 75% to 92% reduction in processing time of packets of interest over the Control configuration, depending on the type of packet. Therefore, the Combined configuration is confirmed to be the best system configuration for minimizing the overall *packet processing time* for all packets entering the system.

When the ability to process SIP packets is added to the system, the mean *packet processing time* required to process non-peer-to-peer packets increases by 115 cycles (0.38 microseconds) and the time required to process BitTorrent Handshake packets increases by 100 cycles (0.33 microseconds). This increase in *packet processing time* is due to the additional software code required to check each packet for the signature of a SIP control packet as well as the signature of a BitTorrent Handshake packet.

One significant finding in this portion of the research is the vastly increased *packet processing time* required for SIP packets over BitTorrent Handshake packets. Using the Combined configuration, the mean *packet processing time* of SIP BYE and INVITE packets of interest is between 99.8 and 115.9 microseconds, versus a mean of 12.9 microseconds for a BitTorrent Handshake packet of interest. There are four main reasons for this large disparity in *packet processing times*.

1. Unlike BitTorrent file info hashes, the sender and receiver SIP URIs are not located at a fixed offset within the packet payload, requiring additional processor time to search the SIP frame contents and find the numbers.
2. When processing SIP packets, the system has to compare both the sender SIP URI and the receiver SIP URI against the list of interest, whereas the system only compares a single file info hash against the list for BitTorrent packets.
3. When performing the list checks, the first 12 bytes of each SIP URI is compared against the list, while only the first 4 bytes of the BitTorrent file info hash are compared.
4. SIP packets are generally much larger than BitTorrent Handshake packets, and thus the system requires more time to copy them. The BitTorrent Handshake frames used in this research are 122 bytes long, while the SIP BYE and INVITE frames are 547 and 963 bytes, respectively.

6.3.2 Analysis of Probability of Packet Intercept Under Load. In the first experiment, the overall goal is to find the configuration that returns the highest *probability of packet intercept* for both non-peer-to-peer and all-peer-to-peer workloads. In the non-peer-to-peer case, where a single BitTorrent packet is sent while the network is under a heavy NETBIOS file transfer load, the Dual Buffer optimization returns a capture rate of over 95%, while the single buffer configurations (with the exception of the Cache configuration, discussed below) all return capture rates of less than 60%. This significant packet loss rate for the single receive buffer configurations is

likely due to the inability of a non-peer-to-peer frame to be processed and cleared from the buffer before the BitTorrent Handshake packet arrives at the system. At 100 Mbps, the mandatory inter-frame gap required by the Ethernet protocol results in a 0.96 microsecond delay between the end of one frame and the beginning of the next. Since the system processes instructions at 300 MHz, it is able to perform at most 300 instructions per microsecond. Therefore, because multiple instructions are required to transfer data from the Ethernet buffer, read the payload contents, and analyze the data, the system cannot keep up with the data flow, resulting in significant packet loss as the system approaches 100% utilization. However, note that the Cache optimization is the exception to this observation; even with a single buffer, enabling the caches results in a capture rate of 96%. This is likely due to the fact that the extremely small processing times provided by the cache enable a packet to be processed in the short interframe time gap.

Adding a second receive buffer to the Ethernet controller dramatically increases the probability of packet intercept under a non-peer-to-peer workload, achieving a 97% capture rate even with no other optimizations incorporated. The use of two receive buffers allows a non-peer-to-peer packet to be processed from one buffer while the BitTorrent Handshake packet is being received in the second buffer. Specifically, the additional buffer provides a minimum of 576 additional bit times ((7-byte preamble + 1-byte delimiter + 64-byte minimum frame size) x 8 bits per byte) for the processing of the non-peer-to-peer frame over the single buffer option [IEE05]. Although this improvement comes at the cost of additional processing cycles, as shown in Table 6.10, the expanded processing window provided by the second buffer more than offsets the cost in individual *packet processing times*. When combined with caching and the improved packet writing scheme, the infrequency of packets of interest, and the small likelihood of traffic saturation on the network link, the final Combined configuration allows the system to successfully capture and process all BitTorrent packets of interest sent into a network with a high non-peer-to-peer traffic load.

For the situation where BitTorrent Handshake packets are sent to the system back-to-back, as in the case of the all-peer-to-peer workload, each system optimization returns a much different *probability of packet intercept*. Using this workload, only the Packet Write optimization results in a greater than 40% *probability of packet intercept*; the other three optimizations all return capture rates of less than 15%. These low capture rates are due to the increased *packet processing time* required for BitTorrent packets over non-peer-to-peer packets. When all four optimizations are used together, the resulting Combined configuration is able to achieve a 100% *probability of packet intercept* for BitTorrent Handshake packets that are received back-to-back by the SUT, which is comparable to the results for the dedicated Wireshark packet sniffer.

One interesting result from this portion of the testing is the fact that the Packet Write optimization returns a *probability of packet intercept* that is four times higher than that of the Cache configuration, yet the *packet processing time* of the Packet Write configuration is over double that of the Cache configuration. To resolve this discrepancy, further testing of the Cache configuration is conducted, and the discovery made that once several hundred packets from the all-peer-to-peer workload are written to the Compact Flash card, approximately one in four packets thereafter suffers from a *packet processing time* of over 125,000 cycles, which is 10 times greater than the average *packet processing time* found in the first experiment. Two possible sources for this increase are the hardware controller that governs Compact Flash write operations and the cache system itself. Regardless of the cause, the occasional increase in *packet processing time* for the Cache configuration results in a lower overall *probability of packet intercept* than the Packet Write configuration, which does not access the cache or the Compact Flash card while the system is actively processing packets.

When the ability to process SIP packets is added to the system, the overall system performance is unchanged when processing single packets of interest under a high non-peer-to-peer network load. Regardless of peer-to-peer packet type (BitTorrent Handshake, SIP INVITE, or SIP BYE), the system returns at least a 97.5% *proba-*

bility of packet intercept. This performance is comparable to that of the Wireshark packet sniffer, which also returned a minimum 97.5% *probability of packet intercept*.

However, when the system is tasked with processing back-to-back peer-to-peer packets arriving at near-line speed, the system's performance depends greatly on the type of peer-to-peer packets arriving at the SUT. For the BitTorrent Handshake packet workload, the system still returns a *probability of packet intercept* of 100%, which is unchanged from the non-SIP processing system. When processing back-to-back SIP BYE packets, the *probability of packet intercept* drops to 90.9%, and when processing back-to-back SIP INVITE packets, the *probability of packet intercept* drops further to 65.8%. These results are expected, since the *packet processing time* of a SIP BYE packet is much greater than that of a BitTorrent Handshake packet, and the *packet processing time* of a SIP INVITE packet is greater than that of a SIP BYE packet. Thus, for the case where the system receives back-to-back packets of interest, the probability of successfully intercepting both packets is at least 90% for BitTorrent Handshake and SIP BYE packets, and is less than two-in-three for SIP INVITE packets. In comparison, the Wireshark packet sniffer achieved a *probability of packet intercept* of greater than 99% for all three packet types, which is likely due to the fact that Wireshark does not perform any extraction or comparison of payload data in the frames it collects.

6.4 Summary

This chapter presents and analyzes the data collected from each of the three tests that were used in the two experiments. A statistical analysis of the performance metrics for each test in each experiment is performed. Then, an overall analysis and discussion of the results from the experiments is provided. The results show that the Combined configuration processes packets of interest 92% faster than the Control configuration, and the probability of intercept for BitTorrent Handshake messages is 99.0%, and the probability of intercept for SIP control packets is 97.6%, under a network traffic load of at least 89.6 Mbps.

VII. Conclusions

This chapter presents the overall conclusions drawn from the research. Section 7.1 compares each research goal with the experimental results and determines if the research objectives have been met. The significance of the research is discussed in Section 7.2. Finally, Section 7.3 provides several recommendations for possible areas of expansion for this research.

7.1 Conclusions of Research

7.1.1 Goal #1: Construct the TRAPP System. The first goal of this research is to construct an FPGA-based system that analyzes traffic on a network, detects a selected peer-to-peer protocol, compares the digital information being shared against a list of interest, and in the case of a match, records selected control frames from the peer-to-peer session in a log file. To accomplish this goal, a Virtex II Pro FPGA system is first modified to accept all network traffic entering its Ethernet controller. The Power PC processor on the board is then programmed with the peer-to-peer detection software, the BitTorrent file info hash-matching algorithm, and the log file writing routine. Finally, a sample list of interest is created and loaded into the board's memory. The entire system is tested against a real-world BitTorrent file transfer, and the successful packet intercept by the system accomplishes the first research goal.

7.1.2 Goal #2: Optimize the System. The second goal of this research is to optimize the system such that it will be able to detect and record all packets of interest on the network, even under a heavy network traffic load. Of the five modifications made to the original system design, the User Alerts modification returns the worst *packet processing times*, increasing the system's processing times between 1,364% and 14,219% when processing BitTorrent packets. The Combined configuration that incorporates three different modifications returns the best *packet processing times*, decreasing the system's processing times between 75% and 92%, depending on the type of packet being processed.

The five modifications made to the original system design return varying changes to the *probability of packet intercept* for packets of interest under a 89.6 Mbps network load, ranging from an increase of 4.4% for the User Alerts to an increase of 89% for the Combined configuration over the original system design, further justifying the Combined configuration's use for the final optimized design. Using the Combined configuration, the optimized system is able to capture a packet of interest with a *probability of packet intercept* of at least 99.0%, using a 95% confidence interval and given an 89.6 Mbps network utilization. This exceeds the hypothesized value of a 95% probability of intercept, proving the hypothesis and meeting the second research goal.

7.1.3 Goal #3: Expand the System. The final goal of this research is to modify the system to accept an additional peer-to-peer protocol with no impact on overall performance. Software enabling the system to process SIP control packets is added, and the entire system is again tested against actual BitTorrent file transfers and actual VoIP phone calls, verifying its capability to intercept real-world data transfers. When the system is modified to accept SIP control packets in addition to BitTorrent Handshake packets, it suffers a minor increase in *packet processing time* for both non-peer-to-peer and BitTorrent Handshake packets. In addition, the variable placement of SIP URIs within SIP control packets leads to an order of magnitude increase in their *packet processing times* over those of BitTorrent Handshake packets.

With one exception, across all types of peer-to-peer packets tested, the updated system is able to capture a packet of interest with a *probability of packet intercept* of at least 97.6%, using a 95% confidence interval and given an 89.6 Mbps network load. However, in the rare case where the updated system is processing packets at a rate of over 94 Mbps, and it receives two SIP control packets in a row, the *probability of packet intercept* decreases to 91% for the second BYE packet and 66% for the second INVITE packet. With the exception of this rare case, these results exceed

the hypothesized value of a 95% probability of intercept, conditionally proving the hypothesis and meeting the final research goal.

7.2 Significance of Research

This research provides the Air Force and other government agencies with a unique method of detecting and tracking both illicit file sharing and VoIP phone call patterns. This system differs from other methods of tracking illicit file sharing in that it is completely passive, meaning the system transmits absolutely no information into the network being monitored, making it completely invisible to users of the network. By designing the system to be completely self-contained on a Virtex II Pro FPGA, the TRAPP system can be easily and inexpensively implemented on any LAN, provided the system has access to a spanning port on the LAN gateway. The simplicity of the system and its FPGA-based implementation enable it to run at very high speeds, ensuring a high probability that a packet of interest is successfully intercepted, even when monitoring a heavily utilized network. Because the tool operates on a spanning port of the network gateway, any failure of the TRAPP system will have no negative impact on the network's performance. Finally, the system can be easily expanded to include additional peer-to-peer protocols with minimum impact on overall system performance.

When fully implemented, the TRAPP system will be an effective weapon in the fight against child pornography. By focusing on those peer-to-peer file transfers that involve the sharing of child pornography, the TRAPP system will give investigators the ability to detect when an illicit file is traversing a network of interest, and allow them to determine from where in the network the transfer is taking place. The system will also provide them with proof, in the form of a BitTorrent data transfer request packet, that the illicit activity is taking place, when it happened, and the address of the computer used for the transfer.

The TRAPP system also has the unique ability to assist in the identification and tracking of terrorist cells similar to those involved in the 2008 Mumbai attacks.

By tracking VoIP phone conversations based on a list of interest containing SIP URIs of known terrorists or terror supporters, TRAPP can determine the SIP URIs with which they are communicating. With these new URIs in hand, authorities can then attempt to track the source of the SIP client and its registrar server, and determine the physical location of the terrorists. In addition, the SIP conversations collected can be aggregated to provide the investigator with a social network of the organization, and may also provide data on the its command and control structure.

7.3 Recommendations for Future Research

The next logical step for this research is to determine how the system performs on a more robust network. Specifically, the TRAPP system should next be tested on a gigabit Ethernet network using a more advanced FPGA board that contains both a faster processor and a gigabit Ethernet controller. The Virtex 5-series FPGA board features a higher processor clock speed of 550 MHz versus 300 MHz for the Virtex II, and contains a gigabit Ethernet controller, making it an excellent choice for the next iteration of the TRAPP system [Xil08a].

Another area for future research is addressing the encryption and obfuscation capabilities of peer-to-peer networks. The next version of TRAPP should be able to detect peer-to-peer control packets that have been obfuscated or encrypted, and the relevant data extracted. However, the entire digital information transfer need not be decrypted; simply knowing what digital information is being transferred or what SIP URIs are being used, which are contained in the protocol's control packets, is sufficient for the stated purpose of the TRAPP system.

This research used a relatively small list of interest size of 1000 entries. Future research should investigate how much larger data sets affect the overall performance of the TRAPP system. However, the data sets will continue to be limited by the size of the FPGA's onboard memory, which necessitated the list size used in the Virtex II-based system.

Finally, the system should be considered for use as a tool for countering the worldwide proliferation of botnets [Ber06]. The TRAPP system could prove to be an effective detection and tracking system for botnet control messages entering and leaving a target network, and this should be investigated as a future capability.

Appendix A. Constructing the System Hardware

This Appendix presents a step-by-step guide to constructing the hardware portion of the FPGA-based forensic tool used in this research. Section A.1 gives an overview of the hardware components used in the construction of the system. Section A.2 provides a step-by-step guide to creating the system using the Xilinx Platform Studio software suite. Section A.3 shows how the Ethernet controller is modified to enable promiscuous mode. Finally, Section A.4 shows how to create the Wireshark-compatible system clock that is used to timestamp frames that are copied to the log file.

A.1 Hardware Description

This section describes the major hardware components used by the TRAPP system. The tool is based on the Xilinx II-Pro Development Board, and all component programming and integration is accomplished using the Xilinx Platform Studio software suite. As shown in Figure A.1, a Power PC core contained on the FPGA executes the code stored in Block RAM modules, and implements those commands on other hardware modules that are connected via the Processor Local Bus (PLB). These components and their functions are detailed below.

A.1.0.1 Microprocessor. The on-chip Power PC 405 processor is used in this system. The design uses the inherent functionality of the Power PC processor to access on board Block RAM, implement control bus functions, and provide the capability for executing software applications.

A.1.1 Block RAM. For this implementation, three 64-kilobyte Block RAM (BRAM) modules are used. The first block is dedicated exclusively to boot up software code, data and instruction memory, and the stack and heap. The second block is dedicated to storage of the lists of interest that are read in from the Compact Flash card. The third block is dedicated to intermediate storage of captured frames prior to their transfer to the log file on the Compact Flash card.

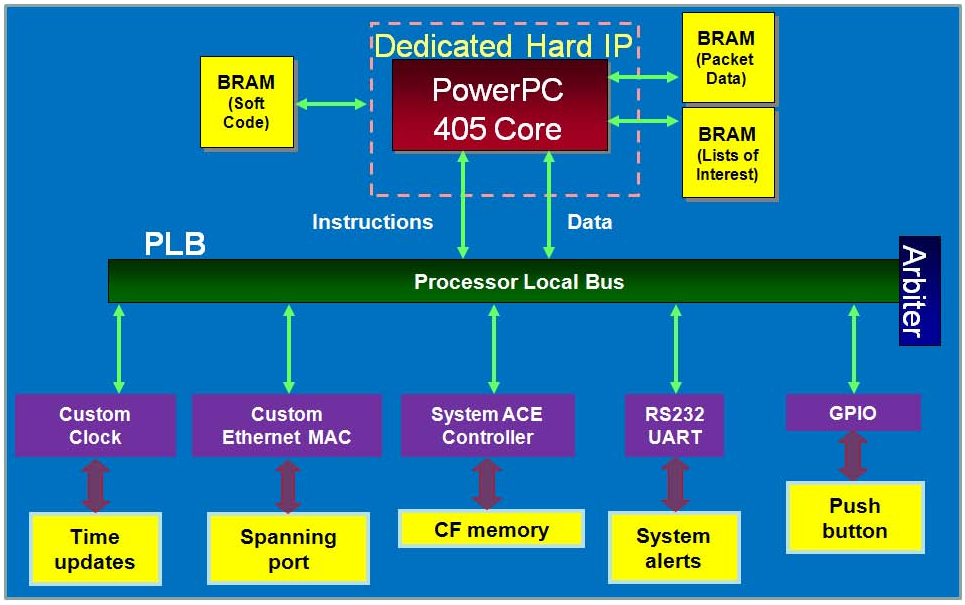

Figure A.1: Hardware Design Block Diagram

A.1.2 XPS EthernetLite Controller. The Intellectual Property (IP) to control the functions of the board's Ethernet connection is provided by Xilinx and is used as the basis for the system's network interface. The VHDL code for this interface is modified to allow packet capture for all packets (promiscuous mode), as the EthernetLite as designed only supports unicast and broadcast traffic. For details on how this Xilinx IP was modified, see Section A.3.

A.1.3 System ACE Controller. The Xilinx-provided IP for the System ACE controller is used to manage and access the Compact Flash removable storage media for storage and retrieval of data. This element is used both to read the lists of interest into the system and to write the packet capture log file.

A.1.4 RS232 UART / General Purpose IO Interfaces. Two interfaces are used to control the functioning of the system and for debugging purposes. Specifically, the terminal interface is used to alert the user when a packet of interest is detected by the system, and to display system messages concerning the transfer of data to and

from the Compact Flash card. The push buttons are used to tell the device to stop execution and to write the log file from the BRAM to the Compact Flash card.

A.1.5 Custom Hardware Clock. This hardware provides the system with a clock that is used to time stamp packets of interest when they are written to the log file. This hardware implementation was added due to the excessive amount of software processing required to generate a Wireshark-compatible time stamp using the PowerPC System Timer.

A.2 Configuring Components on the Virtex FPGA Board

This section provides a step-by-step description of how to create the hardware configuration described above. For this research, Xilinx Platform Studio, release version 10.1.01 is used.

1. To begin, create a new project by selecting "New Project" from the "File" menu. In the resulting pop-up window, shown in Figure A.2, select the "Base System Builder wizard" option and click "OK" to continue.

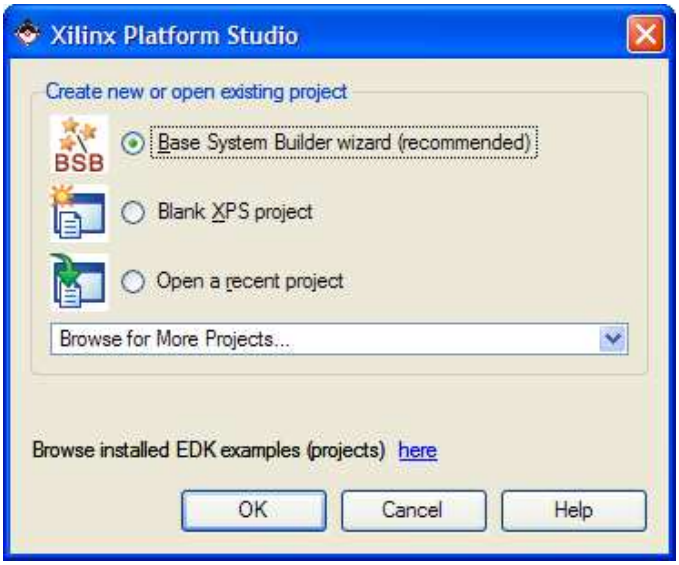

Figure A.2: The Project Creation Options Window

2. In the next pop-up window, shown in Figure A.3, name the project file. In the Advanced Options, check the "Set Project Peripheral Repositories" box, and type the path to the Virtex II Pro library directory contained on the Drivers CD included with the Virtex II Pro board. Then click "OK" to continue.

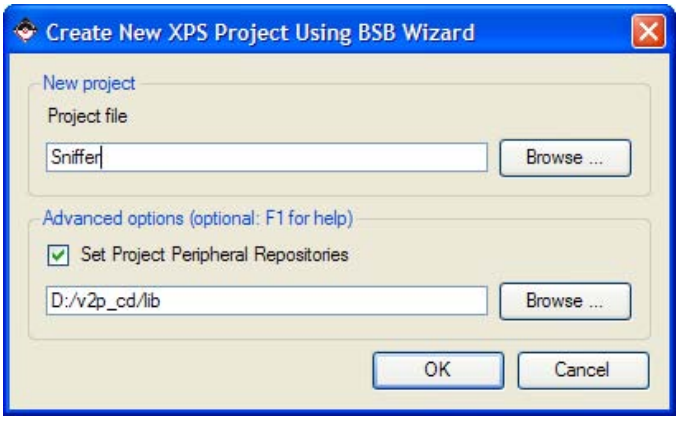

Figure A.3: The Project Creation and Repository Selection Window

3. In the Base System Builder - Welcome window, select "I would like to create a new design" and click "Next" to continue.

4. In the Select Board window, shown in Figure A.4, select “Xilinx” in the Board vendor drop-down menu and “XUP Virtex-II Pro Development System” in the Board name drop-down menu. Ensure that revision C is shown in the Board revision drop-down menu, and click “Next” to continue.

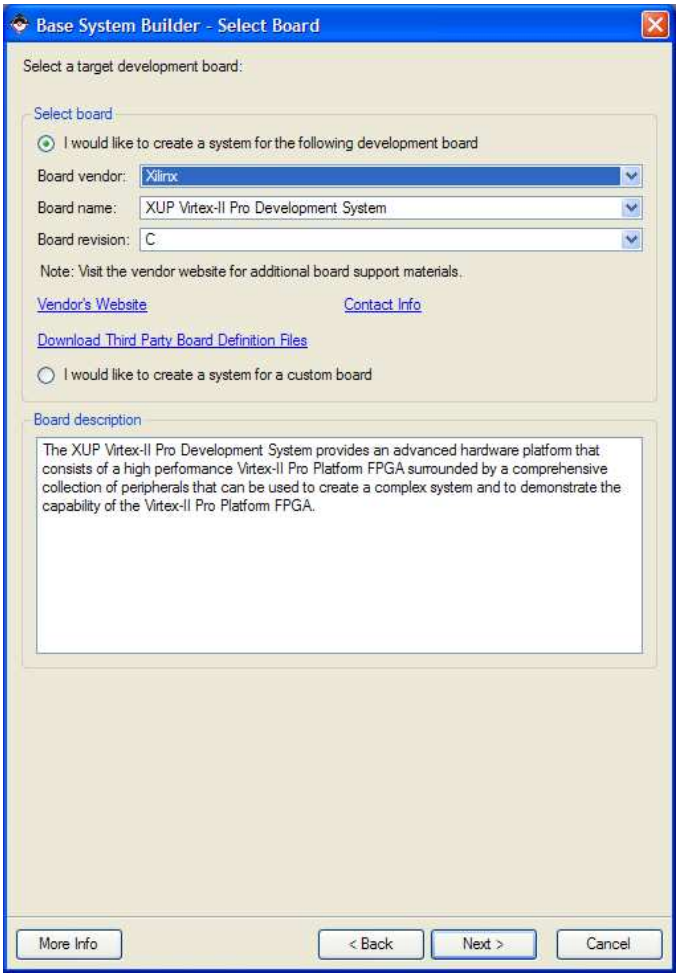

Figure A.4: The Select Board Window

5. In the Base System Builder - Select Processor window, select the “Power PC” radio button, then click “Next” to continue.

6. In the Configure PowerPC Processor window, shown in Figure A.5, select "300.00" MHz for the Processor clock frequency and "100.00" MHz for the Bus clock frequency. Ensure that the "FPGA JTAG" radio button is selected in the Debug I/F section, and that no On-chip memory is selected. Then click "Next" to continue.

Figure A.5: The Configure PowerPC Processor Window

7. In the Configure IO Interfaces (1 of 2) window, shown in Figure A.6, select the boxes for RS232_Uart_1, Ethernet_MAC, and SysACE_CompactFlash. In the RS232 section, select "115200" as the Baudrate. Then click "Next" to continue.

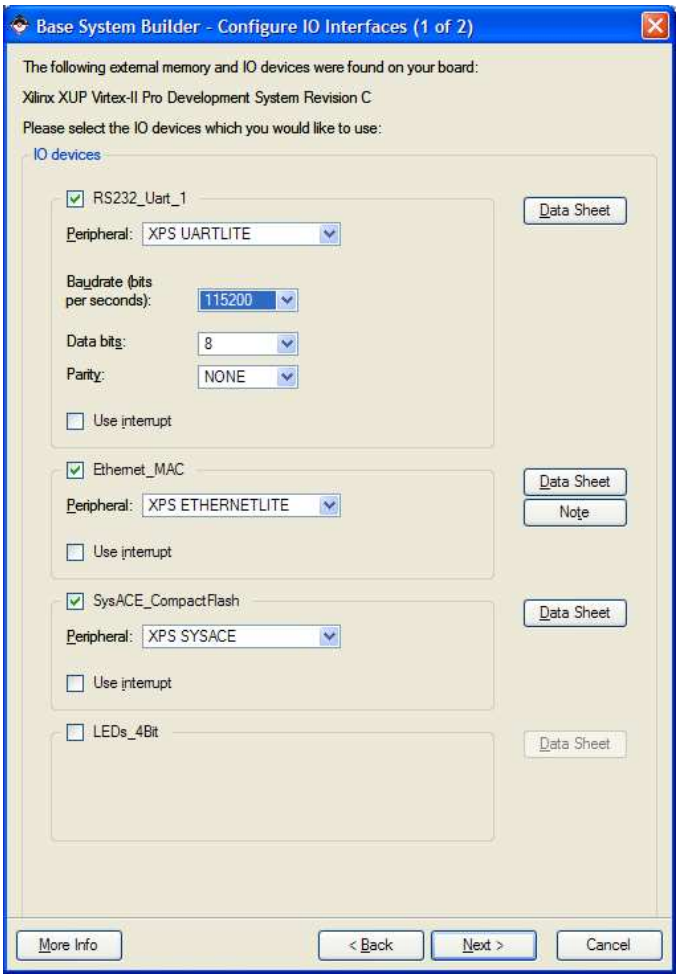

Figure A.6: The Configure IO Interfaces (1 of 2) Window

8. In the Configure IO Interfaces (2 of 2) window, shown in Figure A.7, select the box for PushButtons_5Bit. Then click "Next" to continue.

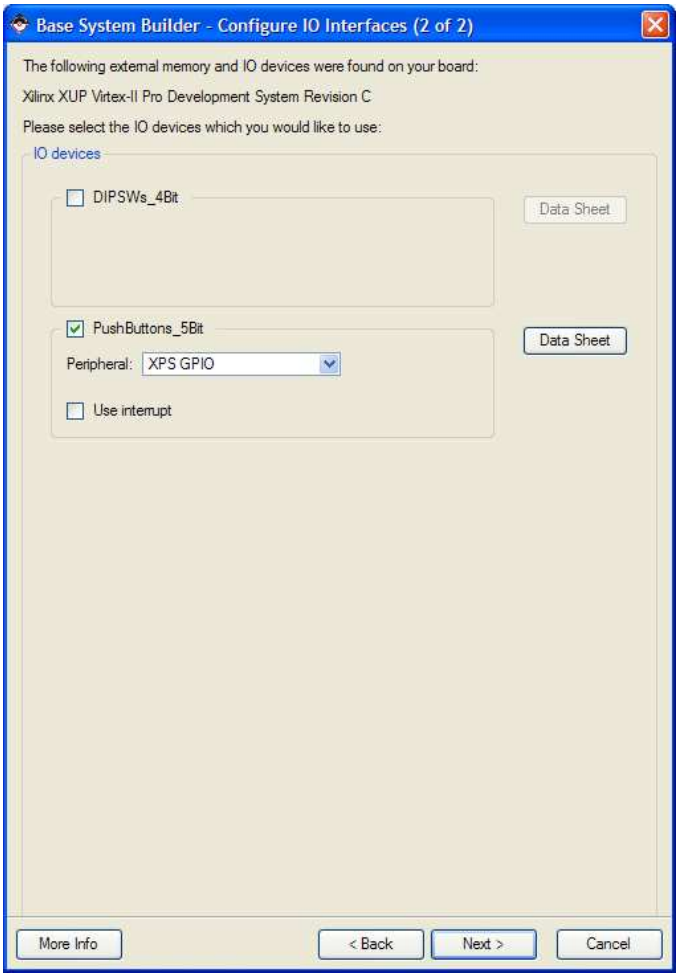

Figure A.7: The Configure IO Interfaces (2 of 2) Window

9. In the Add Internal Peripherals window, shown in Figure A.8, create three BRAM components by clicking the "Add Peripheral" button and selecting XPS BRAM IF CNTLR. For each BRAM, ensure the Memory size is set to 64 Kb.

Figure A.8: The Add Internal Peripherals Window

10. In the Software Setup window, shown in Figure A.9, deselect the Memory test and Peripheral selftest sample application selections, then click "Next" to continue.

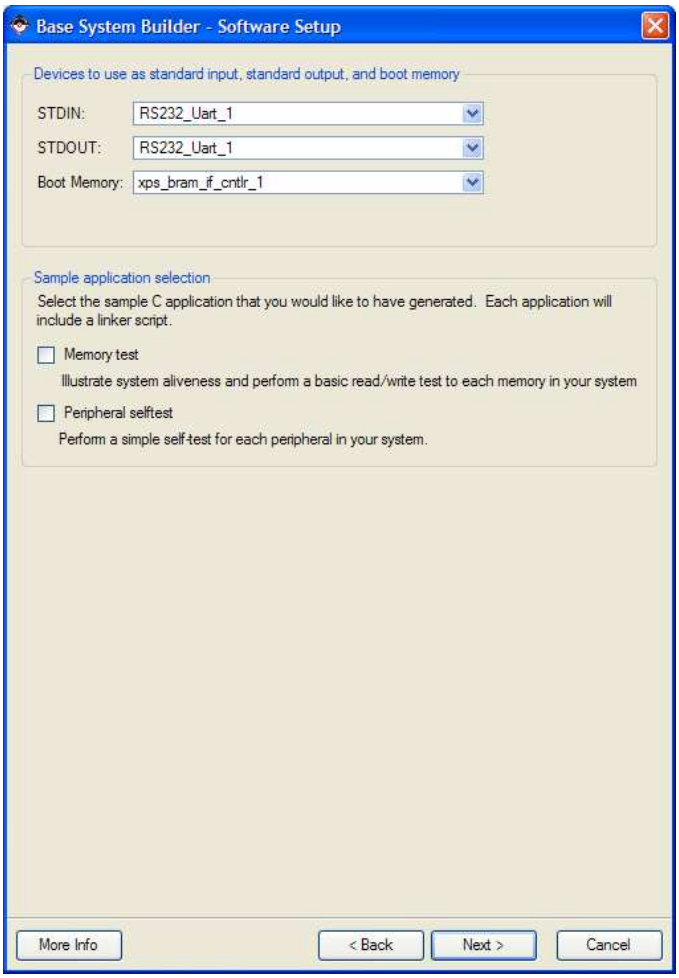

Figure A.9: The Software Setup Window

11. In the System Created window, verify that the appropriate hardware has been added, then click "Generate" to complete the hardware construction.

A.3 Modifying the Ethernet Controller

The first piece of custom IP created for the hardware system is the modification of the Xilinx-provided Ethernet interface to allow it to capture all frames transmitted across the network. Currently, the XPS EthernetLite drivers only support unicast and

broadcast frame reception "out of the box." To force the Ethernet controller to operate in promiscuous mode, the VHDL code in the `EthernetLite's rx_state_2.vhd` file is modified. The file is found in the `C:\Xilinx\10.1\EDK\hw\XilinxProcessorIPLib\pcores\xps_ethernetlite_v2_00_a\hdl\vhdl` directory. The modifications to the file's code are shown in the code segment below:

```
 1  -- Keep these two lines from the original code:
 2  bcastAddrGood <= '1' when checkingBroadcastAdr_i = '1' and
 3  Emac_rx_rd_data_d1(0 to 3) = x"F" else '0'; -- 03-26-04
 4
 5  -- Add the following line to force the controller to think that all
 6  -- non-broadcast frames are unicast:
 7  ucastAddrGood <= '1' when checkingBroadcastAdr_i = '0' else '0';
 8
 9  -- Remove these next two statements...they are no longer needed:
10  -- ucastAddrGood <= '1' when checkingBroadcastAdr_i = '0' and
11  -- (Emac_rx_rd_data_d1(0 to 3) = Mac_addr_ram_data) else '0';--03-26-04
```

In this modification, line 7 is added to set the "Unicast Address Good" variable to '1' regardless of the actual destination MAC address. The '1' value indicates that that the frame is addressed to this Ethernet controller. Lines 10 and 11, which contain the original unicast address comparison logic, are then removed.

Though this modification is deceptively simple, a total deconstruction of the EthernetLite interface driver and receive state machine was required to determine where the modifications should be made. By telling the receive hardware that every packet received is a confirmed unicast frame, the Ethernet controller captures every frame regardless of intended recipient, and passes them up to the software driver.

A.4 Creating the System Clock

The second piece of custom hardware is the development of a custom hardware system clock using VHDL. The clock has two main functions, keeping accurate time in a Wireshark-readable format and updating its clock value when directed by the system software. The core VHDL implementation of the timer is shown in the code segment below:

```
SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  variable counter_usec : STD_LOGIC_VECTOR (0 to 31) := x"00000000";
  variable counter_sec : STD_LOGIC_VECTOR (0 to 31);
  variable count_100 : integer := 0;
  begin
  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
     if Bus2IP_Reset = '1' then
        slv_reg0 <= (others => '0');
        slv_reg1 <= (others => '0');
        slv_reg2 <= (others => '0');
     else
        case slv_reg_write_sel is
          -- If a time index is written to one of the registers from the
          -- software, update the clock secs counter with the new value
          when "100" =>
            -- Read in the time index register to the secs counter
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
              counter_sec(byte_index*8 to byte_index*8+7) :=
                        Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          -- If the clock register has not been written to by the
          -- software, do the following...
```

```
          when others =>
              -- If 100 clock cycles have passed, increment usec counter
              if (count_100 = 100) then
              counter_usec := counter_usec + '1';
              -- Reset the clock tick counter
              count_100 := 0;
              -- If a million usecs have passed, increment sec counter
              if (counter_usec = 1000000) then
                counter_sec := counter_sec + '1';
                -- Reset the usec counter
                counter_usec := x"00000000";
                -- Copy the new secs value to the software register
                slv_reg0 <= counter_sec;
              end if;
              -- Copy the new usecs value to the software register
              slv_reg1 <= counter_usec;
              end if;
              -- Increment the clock tick counter
              count_100 := count_100 + 1;
        end case;
      end if;
    end if;
  end process SLAVE_REG_WRITE_PROC;
```

To provide the system with a Wireshark-readable time output, the timer continuously updates two software-accessible 32-bit registers. One register (`slv_reg0`) returns an unsigned integer representing the number of seconds since system initialization or since the last time update. The other register (`slv_reg1`) returns an unsigned integer representing the number of microseconds since the timer's seconds register was incremented. Note that the timer utilizes the PLB clock, which for this implementation runs at 100 MHz. Thus, the microsecond counter (`counter_usec`)

is incremented by one for every 100 ticks of the bus clock. In addition, the seconds counter (`counter_sec`) is incremented by one for every one million ticks of the microseconds counter. This format matches the Wireshark-standard 64-bit format of a 32-bit unsigned integer for seconds and a 32-bit unsigned integer for microseconds.

To update the system clock with an updated time, the system searches for any Network Time Protocol (NTP) packets being transmitted on the network. When the system finds an NTP packet, it extracts the 32-bit seconds portion of the time stamp and converts it to the Wireshark time format. The software then writes the updated 32-bit seconds value to a designated variable, and sends the 32 bits from the variable via the PLB to the (`counter_sec`) register.

Appendix B. Experimental Data

This Appendix presents the experimental results from the *packet processing time* tests conducted for the two experiments. Section B.1 contains the data tables for tests conducted in the first Experiment. Section B.2 contains the data table for the test conducted in the second Experiment.

B.1 Results of Testing for the BitTorrent Protocol

B.1.1 Non-BitTorrent Packets. Table B.1 below contains the number of cycles required to process a non-BitTorrent packet for each of the six configurations described in the Methodology. For each configuration, 50 packets are sent to the apparatus and the number of CPU cycles required to process the packet are recorded in the table.

Table B.1: Processor Cycles Used to Process a Non-BitTorrent Packet

Control	User Alerts	Packet Write	Dual Buffer	Cache	Optimized
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329
1206	1152	1146	1236	276	278
1206	1152	1146	1452	276	329

B.1.2 Packets with File Info Hash Not On the List. Table B.2 below contains the number of cycles required to process a BitTorrent Handshake packet whose file info hash is not on the list of interest, for each of the six configurations described in the Methodology. For each configuration, 50 packets are sent to the apparatus and the number of CPU cycles required to process the packet are recorded in the table.

Table B.2: Processor Cycles Used to Process a Packet with a Hash Not On the List

Control	User Alerts	Packet Write	Dual Buffer	Cache	Optimized
7296	1045329	7593	7770	1145	1205
7296	1043829	7593	7770	1145	1205
7296	1045329	7593	7770	1145	1205
7296	1043829	7593	7770	1145	1205
7296	1045629	7593	7770	1145	1205
7296	1045479	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1043829	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1043679	7593	7770	1145	1205
7296	1045179	7593	7770	1145	1205
7296	1044879	7593	7770	1145	1205
7296	1044129	7593	7770	1145	1205
7296	1045779	7593	7770	1145	1205
7296	1044129	7593	7770	1145	1205
7296	1043529	7593	7770	1145	1205
7296	1045179	7593	7770	1145	1205
7296	1046079	7593	7770	1145	1205
7296	1043829	7593	7770	1145	1205
7296	1044279	7593	7770	1145	1205
7296	1045629	7593	7770	1145	1205
7296	1043979	7593	7770	1145	1205
7296	1044129	7593	7770	1145	1205
7296	1045329	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1044729	7593	7770	1145	1205
7296	1045179	7593	7770	1145	1205
7296	1045779	7593	7770	1145	1205
7296	1045179	7593	7770	1145	1205
7296	1045929	7593	7770	1145	1205
7296	1045329	7593	7770	1145	1205
7296	1043979	7593	7770	1145	1205
7296	1045479	7593	7770	1145	1205
7296	1043979	7593	7770	1145	1205
7296	1045029	7593	7770	1145	1205
7296	1043979	7593	7770	1145	1205
7296	1043529	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1045479	7593	7770	1145	1205
7296	1044579	7593	7770	1145	1205
7296	1045929	7593	7770	1145	1205
7296	1045029	7593	7770	1145	1205
7296	1044579	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1045629	7593	7770	1145	1205
7296	1044429	7593	7770	1145	1205
7296	1043979	7593	7770	1145	1205
7296	1045029	7593	7770	1145	1205
7296	1045479	7593	7770	1145	1205

B.1.3 Packets with File Info Hash On the List. Table B.3 below contains the number of cycles required to process a BitTorrent Handshake packet whose file info hash is on the list of interest, for each of the six configurations described in the Methodology. For each configuration, 50 packets are sent to the apparatus and the number of CPU cycles required to process the packet are recorded in the table.

Table B.3: Processor Cycles Used to Process a Packet with a Hash On the List

Control	User Alerts	Packet Write	Dual Buffer	Cache	Optimized
104016	1688943	22977	106233	14356	4184
111327	1696977	23607	114813	14216	3773
104016	1690443	22977	106233	13574	3770
124185	1710855	23607	127362	15563	3842
104322	1690026	22977	106539	13613	3770
111633	1697910	23607	115119	14255	3773
104322	1690626	22977	106539	13613	3839
124491	1710588	23607	127668	15527	3773
104628	1689759	22977	106845	13613	3770
111939	1697493	23607	115425	14255	3845
117678	1704441	22977	119628	14819	3740
112245	1697976	23607	115731	14294	3773
104934	1689792	22977	107151	13652	3845
112245	1697976	23607	115731	14294	3773
198645	1787217	22977	200574	22697	3740
112551	1697859	23607	116037	14333	3818
105240	1692075	22977	107457	13691	3740
112551	1698309	23607	116037	14333	3773
118215	1703229	22977	120093	14900	3740
112857	1698192	23607	116343	14372	3812
105546	1691208	22977	107763	13730	3740
112857	1699992	23607	116343	14372	3743
119079	1704264	22977	121086	14978	3809
113163	1700175	23607	116649	14411	3743
105852	1690791	22977	108069	13769	3770
126021	1711203	23607	129198	15599	3812
106158	1691424	22977	108375	13808	3740
113469	1700058	23607	116955	14450	3743
106158	1691874	22977	108375	13808	3815
207222	1794222	23607	210483	22865	3773
106464	1693407	22977	108681	13847	3650
113775	1699791	23607	117261	14489	3848
106464	1693707	22977	108681	13847	3770
126633	1713519	23607	129810	15677	3773
106770	1691640	22977	108987	13886	3770
114081	1699524	23607	117567	14528	3842
120027	1705767	22977	122025	15104	3770
108267	1693011	23607	111753	13787	3773
100956	1686477	22977	103173	13145	3839
108267	1694361	23607	111753	13787	3773
113940	1700628	22977	115821	14348	3770
108573	1694280	23607	112059	13826	3845
101262	1685496	22977	103479	13184	3770
108573	1694430	23607	112059	13826	3773
195387	1782432	22977	197349	21635	3845
108879	1694313	23607	112365	13865	3773
101568	1686129	22977	103785	13223	3770
108879	1694463	23607	112365	13865	3725
114825	1701150	22977	116826	14441	3650
109185	1695846	23607	112671	13904	3653

B.2 Results of Testing Incorporating BitTorrent and SIP

Table B.4 below contains the number of cycles required to process various Peer-to-Peer packets using the final apparatus configuration outlined in Appendix A. A total of 50 packets of each type are sent to the apparatus and the number of CPU cycles required to process the packet are recorded in the table.

Table B.4: Processor Cycles Used to Process BitTorrent and SIP Packets

Not P2P	SIP Not On List	SIP INVITE	SIP BYE	BT Not On List	BT Handshake
595	20108	35534	31031	1544	4331
440	19472	34052	29564	1319	3887
389	19421	34862	29513	1319	3884
440	19421	34490	29834	1319	3956
389	19421	34406	30053	1319	3764
440	19421	33929	29804	1319	3767
389	19421	34229	29933	1319	3803
440	19421	34859	29894	1319	3767
389	19421	34787	29573	1319	3764
440	19472	34859	29423	1319	3806
389	19421	34799	29462	1319	3764
440	19472	34790	29663	1319	3767
389	19472	34745	30002	1319	3866
440	19421	34859	29933	1319	3887
389	19421	34790	30002	1319	3884
440	19421	34862	29933	1319	3962
389	19472	34805	30002	1319	3884
440	19421	34790	29933	1319	3887
389	19421	34862	30002	1319	3884
440	19421	34859	29873	1319	3956
389	19472	34736	30002	1319	3884
440	19421	34862	29873	1319	3857
389	19421	34859	30053	1319	3923
440	19421	34790	29933	1319	3857
389	19472	34808	30002	1319	3884
440	19421	34859	29933	1319	3959
389	19421	34790	30053	1319	3884
440	19421	34862	29984	1319	3857
389	19421	34808	30053	1319	3959
440	19472	34790	29984	1319	3887
389	19472	34862	30053	1319	3953
440	19472	34862	29984	1319	3929
389	19421	34736	30053	1319	3884
440	19472	34862	29984	1319	3857
389	19472	34862	30053	1319	3854
440	19472	34790	29984	1319	3926
389	19421	34805	30053	1319	3854
440	19472	34862	29984	1319	3857
389	19421	34790	30053	1319	3923
440	19421	34859	29984	1319	3887
389	19472	34787	30053	1319	3854
440	19421	34859	29984	1319	3929
389	19421	34859	30002	1319	3854
440	19421	34790	29984	1319	3857
389	19421	34856	30053	1319	3929
440	19421	34859	29984	1319	3887
389	19421	34790	30002	1319	3794
440	19421	34862	29984	1319	3932
389	19421	34856	30053	1319	3884
440	19421	34790	29984	1319	3887

Bibliography

AJ07. Frank Adelstein and Robert A. Joyce. File Marshal: Automatic Extraction of Peer-to-Peer Data. *Digital Investigation*, 4(Supplement 1):43–48, September 2007.

Bas08. Brian Baskin. BitTorrent: The Swarm of Internet Crime, May 2008. PowerPoint presentation available from author and Department of Defense Cyber Crime Center.

Ber06. Scott Berinato. Attack of the Bots. *Wired Magazine*, Issue 14.11, November 2006.

Bit08. BitTorrent Protocol Specification v1.0, May 2008. http://wiki.theory.org/BitTorrentSpecification.

BSCF07. Remi Badonnel, Radu State, Isabelle Chrisment, and Olivier Festor. A Management Platform for Tracking Cyber Predators In Peer-to-Peer Networks. *Proceedings of the Second International Conference on Internet Monitoring and Protection*, page 11, 2007.

CCM+07. K. P. Chow, K. Y. Cheng, L. Y. Man, Pierre K. Y. Lai, Lucas C. K. Hui, C. F. Chong, K. H. Pun, W. W. Tsang, H. W. Chan, and S. M. Yiu. BTM - An Automated Rule-Based BT Monitoring System for Piracy Detection. *Proceedings of the Second International Conference on Internet Monitoring and Protection*, page 2, 2007.

CGD07. W. Q. Cheng, J. Gong, and W. Ding. Identifying BT-like P2P Traffic by the Discreteness of Remote Hosts. *Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 237–238, 2007.

Cis02. Cisco. Cisco Introduces New SIP-enabled Voice over IP Solutions, March 2002. http://newsroom.cisco.com/dlls/prod_031102.html.

Coh03. Bram Cohen. Incentives Build Robustness in BitTorrent, May 2003. http://www.bittorrent.org/bittorrentecon.pdf.

Coh08. Bram Cohen. The BitTorrent Protocol Specification, February 2008. http://www.bittorrent.org/beps/bep_0003.html.

Cor05. CounterPath Corporation. Xten Softphone SDK Delivers PC-to-PC VoIP in New Yahoo! Messenger, June 2005. http://www.counterpath.com/xten-softphone-sdk-delivers-pc-to-pc-voip-in-new-yahoo-messenger.html.

Cor08. CounterPath Corporation. X-Lite VoIP Softphone, September 2008. http://www.counterpath.com/x-lite.html&active=4.

Dir07. Torrent Directory. Terminology of BitTorrent, July 2007. http://www.torrentdirectory.org/articles/article-2.html.

DS08a. Bill Dedman and Bob Sullivan. GFR/CopyRouter Process Flow, October 2008. http://msnbcmedia.msn.com/i/msnbc/Sections/NEWS/PDFs/081016_copyrouter.pdf.

DS08b. Bill Dedman and Bob Sullivan. ISPs are Pressed to Become Child Porn Cops, October 2008. http://www.msnbc.msn.com/id/27198621.

DVR07. Hamza Dahmouni, Sandrine Vaton, and David Rosse. A Markovian Signature-Based Approach to IP Traffic Classification. *Proceedings of the 3rd Annual ACM Workshop on Mining Network Data*, pages 29–34, June 2007.

EAM06. Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic Classification Using Clustering Algorithms. *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, pages 281–286, September 2006.

Fel04. Geoff Fellows. Peer-to-peer Networking Issues-An Overview. *Digital Investigation,*, pages 3–6, February 2004.

FIP93. FIPS 180-1 - Secure Hash Standard, May 1993. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

Gil08. Paul Gil. "Peer Guardian" Firewall: Keep Your P2P Private, January 2008. http://netforbeginners.about.com/od/peersharing/a/peerguardian.htm.

Gon05. Yiming Gong. Identifying P2P Users Using Traffic Analysis, July 2005. http://www.securityfocus.com/infocus/1843.

Goo08. Google. Google Talk for Developers, October 2008. http://code.google.com/apis/talk/open_communications.html.

GPW06. Matthew Gebski, Alex Penev, and Raymond Wong. Protocol Identification of Encrypted Network Traffic. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 957–960, December 2006.

Hpi08. Hping. Hping - Active Network Security Tool, July 2008. http://www.hping.org/.

IEE05. IEEE. IEEE Standard 802.3, December 2005. http://standards.ieee.org.

Kah08. Jeremy Kahn. Mumbai Terrorists Relied on New Technology for Attacks, December 2008. http://www.nytimes.com/2008/12/09/world/asia/09mumbai.html?_r=1.

KBFC04. Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport Layer Identification of P2P Traffic. *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 121–134, 2004.

KPF05. Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 229–240, October 2005.

Les08a. Lawrence Lessig. Free Culture, June 2008. http://beta.legaltorrents.com/torrents/20-lawrence-lessig—free-culture.

Les08b. Lawrence Lessig. Free Culture (Audio Book), June 2008. http://beta.legaltorrents.com/torrents/19-lawrence-lessig—free-culture.

Mac06. Richard MacManus. The Underground World of Private P2P Networks, August 2006. http://www.readwriteweb.com/archives/private_p2p.php.

MRR08. Florian Mendel, Christian Rechberger, and Vincent Rijmen. Secure Enough? Re-Assessment of the Worlds Most-Used Hash Function, June 2008. http://www.isgtw.org/?pid=1000711.

MW06. Alok Madhukar and Carey Williamson. A Longitudinal Study of P2P Traffic Classification. *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 179–188, September 2006.

Off05. Government Accounting Office. File Sharing Programs: The Use of Peer-to-Peer Networks to Access Pornography, May 2005. http://www.gao.gov/new.items/d05634.pdf.

oJ08. United States Department of Justice. RCFL Program Annual Report for Fiscal Year 2007, 2008. http://www.rcfl.gov/downloads/documents/RCFL_Nat_Annual07.pdf.

Owe08. Glen Owen. Taliban Using Skype Phones to Dodge MI6, September 2008. http://www.dailymail.co.uk/news/worldnews/article-1055611/Taliban-using-Skype-phones-dodge-MI6.html.

P2P07. P2P Traffic Is Booming, BitTorrent the Dominant Protocol, November 2007. http://torrentfreak.com/p2p-traffic-still-booming-071128/.

Plo00. Dave Plonka. University of Wisconsin-Madison, Napster Traffic Measurement, March 2000. http://net.doit.wisc.edu/data/Napster/.

Pro06. Rice University WARP Project. Wireless Open-Access Research Platform, June 2006. http://warp.rice.edu/trac/browser/PlatformSupport/WARPMAC/warpmac.c.

Pro08a. Tera Term Pro. Tera Term Pro Terminal Emulator, July 2008. http://hp.vector.co.jp/authors/VA002416/teraterm.html.

Pro08b. Protocol Obfuscation, May 2008. http://wiki.emule-web.de/index.php/Protocol_obfuscation.

RE08. Remote-Exploit. BackTrack, June 2008. http://www.remote-exploit.org/backtrack_download.html.

Rea05. Marguerite Reardon. Ups and Downs of Consumer Broadband, August 2005. http://news.cnet.com/Ups-and-downs-of-consumer-broadband/2100-1034_3-5810534.html.

RFC94. RFC 1738 - Uniform Resource Locators (URL), December 1994. http://www.ietf.org/rfc/rfc1738.txt.

RFC99. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

RFC01. RFC 3174 - US Secure Hash Algorithm 1 (SHA1), September 2001. http://www.faqs.org/rfcs/rfc3174.html.

RFC02. RFC 3261 - SIP: Session Initiation Protocol, June 2002. http://www.faqs.org/rfcs/rfc3261.html.

SGD+02. Stefan Saroiu, Krishna Gummadi, Richard Dunn, Steven Gribble, and Henry Levy. An Analysis of Internet Content Delivery Systems. *ACM SIGOPS Operating Systems Review*, 36:315–327, 2002.

Spe08. Special Applications Port List, May 2008. http://www.practicallynetworked.com/sharing/app_port_list.htm.

SSW04. Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. *Proceedings of the 13th International Conference on World Wide Web*, pages 512–521, May 2004.

Sve07. Peter Svensson. Comcast Blocks Some Internet Traffic, October 2007. http://www.msnbc.msn.com/id/21376597/.

The06. The 'One Third of All Internet Traffic' Myth, September 2006. http://torrentfreak.com/bittorrent-the-one-third-of-all-internet-traffic-myth.

The08. The Gnutella Protocol Specification v0.4, May 2008. www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

Tho05. Clive Thompson. The BitTorrent Effect. *Wired Magazine*, Issue 13.01, January 2005.

Tri08. Trixbox. Trixbox, an Asterisk-based PBX Phone System, September 2008. http://www.trixbox.org/.

Tys08. Jeff Tyson. How the Old Napster Worked, June 2008. http://computer.howstuffworks.com/napster2.htm.

Ubi08. Ubiquity. Understanding SIP, July 2008. http://www.sipcenter.com/sip.nsf/html/WEBB5YNVK8/$File/Ubiquity_SIP_Overview.pdf.

uTo08. uTorrent. uTorrent - The Lightweight and Efficient BitTorrent Client, June 2008. http://www.utorrent.com/.

VMW08. VMWare. VMware Player, September 2008. http://www.vmware.com/products/player/.

Wir08. Wireshark. Wireshark Network Protocol Analyzer, July 2008. http://www.wireshark.org/.

WMM06. Charles Wright, Fabian Monrose, and Gerald Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *The Journal of Machine Learning Research*, 7:2745–2769, December 2006.

Xil08a. Xilinx. Virtex-5 Family Overview, August 2008. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

Xil08b. Xilinx. Xilinx University Program Virtex-II Pro Development System, June 2008. http://www.xilinx.com/products/devkits/XUPV2P.htm.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2009	Master's Thesis	Sept 2007 — Mar 2009

4. TITLE AND SUBTITLE

An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Karl R. Schrader, Maj, USAF

5d. PROJECT NUMBER

ENG09 310

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/09-10

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Information Operations Center
Attn: Mr. Robert J. Kaufman
102 Hall Boulevard, Suite 345
San Antonio, TX 78243
(210) 977–5377; robert.kaufman@lackland.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFIOC/IO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research addresses the problem of tracking digital information that is shared using peer-to-peer file transfer and VoIP protocols for the purposes of illicitly disseminating sensitive government information and for covert communication by terrorist cells or criminal organizations. A digital forensic tool is created that searches a network for peer-to-peer control messages, extracts the unique identifier of the file or phone number being used, and compares it against a list of known contraband files or phone numbers. If the identifier is on the list, the control packet is saved for later forensic analysis. The system is implemented using an FPGA-based embedded software application, and processes file transfers using the BitTorrent protocol and VoIP phone calls made using the Session Initiation Protocol (SIP). Results show that the final design processes peer-to-peer packets of interest 92% faster than a software-only configuration, and is able to successfully capture and process BitTorrent Handshake messages with a probability of at least 99.0% and SIP control packets with a probability of at least 97.6% under a network traffic load of at least 89.6 Mbps.

15. SUBJECT TERMS

computer networks, peer-to-peer networking, information security, criminal investigations, forensic analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins
U	U	U	UU	164	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636 x7979; barry.mullins@afit.edu